8-2019

Design of a CMOS-Memristive Mixed-Signal Neuromorphic System with Energy and Area Efficiency in System Level Applications

Gangotree Chakma
University of Tennessee, gchakma@vols.utk.edu

Design of a CMOS-Memristive Mixed-Signal Neuromorphic System with Energy and Area Efficiency in System Level Applications

A Dissertation Presented for the

Doctor of Philosophy

Degree

The University of Tennessee, Knoxville

Gangotree Chakma

August 2019

To my wonderful parents who are my continuous source of inspiration and enthusiasm.
Thank you for believing in me and nurturing my soul with your love and encouragement.

Acknowledgments

I would like to thank my advisor Dr. Garrett Steven Rose for the patient guidance, encouragement and advice he has provided throughout my graduate studies. Without his guidance and persistent help this dissertation would not have been possible.

I would also like to thank Dr. Mark Dean and Dr. James Plank to inspire me sharing insightful thoughts and ideas on different projects. Moreover, I am grateful to them for taking time for serving on my PhD committee.

A special thanks goes to Dr. Hugh Medal to show interest in learning more about Neuromorphic Computing and being one of my PhD committee members.

I would also like to thank Dr. Catherine Schuman from the bottom of my heart for always being a mentor to me and guiding me through my research and advising me as I march forward.

I am thankful to the Department of Electrical Engineering and Computer Science, UTK and all the staff members for their excellent support in administrative works and logistic sides. They have been a constant support from the beginning of my admission till date.

I am also thankful to the office of the Chancellor for awarding me the Chancellor's fellowship which helped me financially throughout my graduate studies.

Next, I would like to share my appreciation for the constant support and assistance from my fellow lab-mates Mesbah Uddin, Md Badruddoza Majumder, Md Musabbir Adnan, Sherif Amer, Sagarverma Sayyaparaju, Ryan Weiss, Nicholas Skuda and Samuel Brown. I enjoyed working with them on different projects and discussing interesting ideas.

Lastly, I would like to finish expressing my gratitude to my wonderful parents, Tapan Bikash Chakma and Jolma Chakma, my lovely younger sister Padmasree Chakma and my dearest friends H M Ashfiqul Hamid, Sadika Amreen, Nameera Tahseen, Dhara Islam Mallik

and Muhammad Redwan Hassan for their support and encouragement. I could not have been successful without the suggestion and assistance from all of these excellent people.

Abstract

The von Neumann architecture has been the backbone of modern computers for several years. This computational framework is popular because it defines an easy, simple and cheap design for the processing unit and memory. Unfortunately, this architecture faces a huge bottleneck going forward since complexity in computations now demands increased parallelism and this architecture is not efficient at parallel processing. Moreover, the post-Moore's law era brings a constant demand for energy-efficient computing with fewer resources and less area. Hence, researchers are interested in establishing alternatives to the von Neumann architecture and neuromorphic computing is one of the few aspiring computing architectures that contributes to this research effectively. Initially, neuromorphic computing attracted attention because of the parallelism found in the bio-inspired networks and they were interested in leveraging this advantage on a single chip. Moreover, the need for speed in real time performance also escalated the popularity of neuromorphic computing and different research groups started working on hardware implementations of neural networks. Also, neuroscience is consistently building a better understanding of biological networks that provides opportunities for bridging the gap between biological neuronal activities and artificial neural networks. As a consequence, the idea behind neuromorphic computing has continued to gain in popularity. In this research, a memristive neuromorphic system for improved power and area efficiency has been presented. This particular implementation introduces a mixed-signal platform to implement neural networks in a synchronous way. In addition to mixed-signal design, a nano-scale memristive device has been introduced that provides power and area efficiency for the overall system. The system design also includes synchronous digital long term plasticity (DLTP), an online learning methodology that helps train the neural networks during the operation phase, improving the efficiency in learning

when considering power consumption and area overhead. This research also proposes a stochastic neuron design with a sigmoidal firing rate. The design introduces variability in the membrane capacitance to reach different membrane potential leading to a variable stochastic firing rate.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

The human brain is a wonderful creation of nature that possesses the ability to complete numerous amounts of complex calculations within a fraction of a second. All of these complex computations are the result of transmitting data using electro-chemical signals as these data are transmitted from one neuron to another. The human brain contains hundreds of billions of neurons which constitute the computing cores of the brain with each neuron and interconnected to others via highly efficient interconnection wires referred as synaptic weights or synaptic connections. The power or strength of any transmitted signal depends on the synaptic weights of the interconnects. If the synaptic weight is high, the transmitted signal from one neuron to another would be more powerful as it is propagated through the synapse. Each neuron receives the weighted signals from multiple synapses and stores the summed charge of the incoming signals from preceding neurons. Once the stored charge exceeds the threshold of the neuron the neuron transmits an output signal to the succeeding neurons. This condition is known as the firing of a neuron.

One of the interesting features of the human brain is its cognitive ability. Since artificial neural networks are inspired by the human brain, the architecture should preserve cognitive features such as an ability to acquire knowledge from the surroundings. This knowledge transfer can be translated as the ability to adapt to different outputs while performing tasks like image and speech recognition. This adaptation is completed gradually through the

learning process which is similar to cognitive learning. During learning, the synaptic weights are updated based on rewards or punishments. That's how the information is transferred from one neuron to the other in neural network.

Drawing inspiration from the complex computations and learning processes in biological neural networks, several computational algorithms have been developed. Artificial Neural Networks (ANNs) are one of the most interesting network platforms that mimic biological neural networks. In an ANN, there are artificial neurons which are similar but not the same as biological neurons. An ANN also contains synapses for transmitting weighted signals from one neuron to another. Mostly, an ANN is a mathematical model for how biological neural networks process information. Hence, it is very popular in tasks related to image classification and speech recognition. These complex tasks mostly depend on the existing von Neumann architecture. Therefore, ANN computations are less efficient as compared to biological counterparts. In the human brain the biological neural network mostly functions like a large parallel machine. On the other hand, ANNs rely on sequential machines where almost all the information needs to be processed in a queue.

In order to increase computing efficiency, there is a need for parallel processing of the ANNs. As a result, researchers have been looking for alternative computing options rather than simply using the conventional von Neumann computing architecture. Moreover, the research in specialized hardware for ANNs has become an exciting research sector. This hardware specialized with parallel processing for ANNs can be noted as neuromorphic circuits. In the literature, there have been interesting works on neuromorphic computing from the early 50s to recent decades. There are numerous contributions on neuromorphic computing hardware. Some of these are digital in nature as in [89] while some follow an analog approach [57, 83, 84]. When these systems are compared against one another, the digital implementations are found to be more robust, scalable and noise tolerant especially in terms of network communication. However, digital approaches are more area intensive [57]. On the other hand, analog implementations are more area and energy efficient with less silicon area and processing speed. But there are disadvantages of using capacitors to hold synaptic weights [84] or resistors to represent synaptic connections [36], resulting in poor

area and energy efficiency. In [68], several existing implementations of neural networks have been discussed in detail.

According to the literature, Moore's law has come to a saturated stage and hence the semiconductor industry has been experiencing a significant slowdown in performance. Moving toward lower technology nodes might help in reducing area but as of late it is not contributing immensely in increasing the computing speed. Moreover, other limiting factors such as power consumption and architectural limitations also have an effect on the performance of the computing machines. The research presented here aims to contribute in overcoming these limitations by leveraging alternative computing systems, specifically neuromorphic computing. In addition, this work utilizes emerging nano-scale devices, specifically the memristor, to overcome power and size challenges.The proposed system leverages a Spiking Neural Network (SNN) architecture to build a platform for neuromorphic computing [60].

1.2 Research Goal and Contribution

1.2.1 Research Goal

Since neuromorphic computing can be defined as one of the fields to help achieve Moore's law maintain it's activity, it is exciting to work in many different fields of neuromorphic computing. As mentioned earlier, neuromorphic computing is an area of research where researchers starting from neuroscience and mathematics to circuit design work together but with very different perspectives. So, our research goal is to try and build a neuromorphic system for society at large from a high to low-level point of view. Thus, we collaborate with people from the algorithmic level to build a framework and translate the architecture to low-level circuit design. This way we can help the community by providing a complete software-hardware system. In order to do that, we start with [86] where Schuman *et. al* introduce a software framework for a spiking neural network architecture which could provide very sparse networks for a variety of applications. This approach is also capable of online learning during and run-time. The architecture is interesting from a system level perspective

because the framework could be helpful in designing energy efficient networks where the networks generated are themselves sparse. In addition, the online learning mechanism can be translated into low-level circuit designs which would provide an interesting way to build a neuromorphic system. Moreover, we had collaborators from device physics level who helped us in providing some experimental data of memristors that showed promises to be used as a part of the system to ensure energy and area efficiency.

1.2.2 Research Contribution

Starting with the high-level architecture known as neuroscience-inspired dynamic architectures (NIDA) [85], we took a very detailed look at the components available in the high-level architecture and re-create them in the implementation of a memristor-based system. So, we started from analyzing different high-level networks and their functionalism. We got the details of how NIDA works and how we can optimize the hardware so that it follows NIDA. Interestingly, we found out that NIDA is asynchronous in nature, whereas we were looking at mixed-signal design which involves synchronous designs. So, our target was to use similar encoding of inputs to both the high and low-level system keeping the core functionality similar. My research contribution to this project is as follows.

- Initially, my research began with the design of a synapse for a neuromorphic system that uses memristors for the main synaptic component. The design includes two memristors connected to each other in a back to back manner. This twin memristive synapse is capable of producing both positive and negative synaptic weights depending on the incoming current directions. Apart from the memristors, there are some digital blocks that help establish the online learning mechanism. I analyzed the energy consumption of each synapse to determine that it was low relative to existing designs. The synapse design is detailed in chapter *three*.

- One significant component of my research is the design of an integrate and fire neuron (IAF). Like other IAF neurons, this neuron also accumulates charges from the incoming synaptic inputs and then generates a pulse whenever the accumulated charge crosses a threshold. The interesting part of this design is the output from the neuron is a

digital pulse. The reason behind mentioning this is because we wanted a system which can leverage the beauty of analog computation in the core alongside the efficiency and robustness of digital communication from the outside. Hence, I designed the neuron to perform all core computation in analog and then transfer the signals from one layer of neurons to others. To be consistent with the high-level architecture, the neuron is also designed to assist with the long term potentiation and long tern depression mechanisms which are elaborately discussed in chapter *three*. Thus, the overall contribution here is the design of a mixed-signal neuron which features online learning and efficient analog computation with robust digital communication.

- Combining the stated synapse and neuron design, I helped in designing a neuromorphic core that contains a neuron and several synapses. Unlike the crossbar architecture, this architecture works as a core itself and the computation can be done locally before being connected to the global system. Since we could translate the networks from NIDA to a hardware level, we obtained reasonable accuracy and energy estimates for different application classes, including classification and control. To obtain the total energy estimate, I analyzed the neuron and synapse models to classify their energy consumption in energy per spike criteria. Then we can obtain an energy estimate when provided with the activity factors from high-level simulation results.

- Lastly, I have added a new feature in the neuron design that provides stochasticity. The stochastic effect is added because noise is an important feature in biological neurons. Thus, I present a stochastic version of the IAF neuron design using capacitive variance. The results from this research is included in chapter *six*. As a proof of concept, we can see that the neuron has a probabilistic firing rate depending on the number of input pulses. Results are provided for a shape recognition network using deterministic and stochastic versions of the neuron. The results present the advantages of stochastic neurons over deterministic ones in order to analyze noisy images. Also, the energy consumption was shown to be in a similar range that of deterministic one. Thus, the stochastic design is also energy-efficient.

1.3 Overview of Dissertation

This dissertation is spread over *seven* chapters. Chapter *one* sets up the motivation behind the neuromorphic research with particular research goals and contributions. Previous works on different neuromorphic computing architectures are described in chapter *two*. In Chapters $three - four$, the design of the proposed neuromorphic architecture is detailed with extensive description, where chapter *three* describes the memristive synapse and explains the construction of mixed signal neurons. Chapter $four$ presents the neuromorphic system integrating the pieces and also shows the software framework used for the system level design. Results from energy analyses of the whole system for different applications are discussed in chapter $five$. Chapter *six* proposes a novel design of introducing stochasticty in neurons. Chapter *seven* concludes the dissertation and provides future work suggestion where few directions are highlighted for leveraging the proposed design in different interesting applications.

Chapter 2

Related Work on Neuromorphic Computing

2.1 Related Work on Synapse Design

In neuromorphic architecture, the synapse is the connector from one neuron to another. It stores a synaptic weight and, in relation to axonal delays, it can also store delay information for the speed of spikes traveling from neuron to neuron. Researchers have developed several synapse models with several features inspired by biology.

Some synapse designs are more interested in modelling the ion pumps found in nature [35] while some are more interested in modelling the ion channels [72]. Researchers have also shown success in implementing the spike time dependent plasticity (STDP) model for learning in biological synapses [23]. This STDP mechanism is one of the popular learning algorithms for spiking neural networks. However, if we want to consider non-spiking networks, other approaches include convolutional neural network [34], winner-take-all circuit [74] and some also learning rules such as back propagation [30] and least mean square [96].

Considering the implementation and the devices used in designing synaptic hardware, we can find a good amount of variety from static CMOS design and also emerging devices. CMOS has been a popular choice from the very start because CMOS technology has been well-established and is relatively easy to design and fabricate. In [42], authors have designed a CMOS synapse with a 0.8 μm CMOS process and achieved both short and long term

plasticity for the synapses. It contains four different stages for the synapse design, including STDP mechanism, STD, bi-stability and a current mirror circuit to generate inputs to the neurons.

There are more works in the literature such as [28, 51, 100] where CMOS has been used in the design of synapses. For instance, authors in [51] implemented a synapse design with fully analog components leveraging a 0.6 μm CMOS process technology. This work contains two operational transconductance amplifiers (OTAs) to replicate the synaptic weights and also includes on-chip STDP learning. The synapse architecture described in [28] provides a very similar approach. However, [28] interestingly introduced a crossbar structure of memristors to reduce the size of the analog CMOS synapse design.

Memristors are first proposed by Chua in 1971 [22] as a theoretical circuit component. Later HP lab fabricated their own memristor in 2008 [113]. Memristors are one of the most promising emerging devices in neuromorphic computing because they exhibit some characteristics that can be found in biological synapses, such as the STDP mechanism. Moreover, memristors are non-volatile and nano-scale devices that make them viable for designing area and energy efficient systems. Also, with the saturation of Moore's law, it has become critical to work with non-linear CMOS technologies in designing vast neuromorphic systems. Hence, leveraging memristors researchers have proposed several synapse designs. Some of them, such as [40, 5] use the memristive crossbar design to implement neuromorphic synapses. The primary advantage of using crossbars is that a high density of synapses can be reached using the crossbar architecture. Moreover, physical crossbars have been fabricated to prove the efficiency of the architecture. There are other architectures such as [49] where the memristor bridge synapse idea has been proposed to represent both positive and negative weights. This structure uses four memristors connected as a Whitstone bridge connection with the input voltage direction deciding the weight orientation.

Other than memristors, there are some other interesting materials used in designing synaptic components such as floating gate transistors, spin devices and phase change memories. Floating gate transistors are mainly used as flash memory devices [117] to provide synaptic weight storage and also implement the STDP mechanism [79]. On the other hand,

both phase change memory [107] and Spintronic devices [106] are used for their high density and implementation of learning behaviors.

2.2 Related Work on Neuron Design

Biological neurons transmit signals using complex chemical processes in which the release of neurotransmitters modulates the electrical potential of individual neurons [68]. When looking at the spiking neuron as a core building block of ANNs, at its most basic it can be modeled by a comparator circuit that compares an input voltage to a pre-defined threshold and if the input is over the threshold, it generates a voltage spike as output (i.e. a voltage pulse with a fixed pulse width is generated). As long as the input voltage remains above the threshold, the circuit will continue spiking. In biological systems these spikes typically have a frequency on the order of milliseconds. Many designs maintain this firing rate in order to mimic biological neurons as closely as possible, though some proposed circuits operate in accelerated time. Here are a number of approaches to modeling neurons that attempt to replicate this spiking behavior with varying degrees of biological accuracy. The most common are the Hodgkin-Huxley model [37], the Izhikevich Model [44], and the Leaky Integrate and Fire model [1]. Among these three, the Hodgkin-Huxley neuron models biological behavior more closely and emulates the biochemical processes. It allows researchers to study brain functionality in detailed manner and hence helps in implementing the brain features in hardware with precision. The drawback of this model is that it can cost high power and chip area consumption [16]. The next model is an updated version of Hodgkin-Huxley. The Izhikevich model [44] is comparably easier to implement because it compromises the biological function with simpler circuits. So, it can achieve better energy and area efficiency in hardware implementations. The third model is the Leaky Integrate and Fire (LIF) neuron model, mentioned by Carver Mead in [62]. Mead described an axon-hillock circuit to represent the mechanism of LIF. In the axon-hillock circuit, an amplifier is used to generate spike events. An input current is used to charge a capacitor, which represents the neural circuits membrane capacitance, until the switching threshold is reached and the output moves to VDD (power rail voltage). Once a spike is generated, a feedback circuit is used

to discharge the membrane capacitor and cause the amplifier to switch back to ground. In its most straight forward implementation this circuit uses a basic two-inverter amplifier and the neurons threshold voltage is entirely dependent on the switching characteristics of the transistors being used to implement it. This implementation is the basis of working with a simpler circuit for neuron representation.

There have been different implementations of Integrate and Fire (IAF) neurons. In [43], a design of a conductance based silicon neuron has been introduced. Here the neuron is implemented as a current mode conductance based neuron with plasticity. The output current here is proportional to the injected spikes which is analogous to the integrate and fire mechanism. Thus, this silicon neuron is a good representation of IAF. Another neuron described in [110] is a pretty good example of the IAF neuron. This neuron has a low-power op amp operating in two asynchronous phases. First one is the integration phase and the next is the firing phase. During the integration phase the op amp acts as a leaky integrator with a preferred leak rate and charges a capacitor based on the incoming input spikes. While charging the capacitor, the membrane potential gradually increases upto a certain voltage which is called the threshold. When the membrane potential exceeds the threshold, the op amp enters the firing phase and acts as a buffer to propagate the input spikes in the forward direction and the output spikes to the synapse inputs.

Usually, most of the available neuron implementations are pure CMOS silicon neurons. However, there are other emerging materials which are being used in designing neurons because of their efficiency in energy consumption and area optimization. For instance, memristors are being used in the neurons to define stochastic nature and define complex spiking behavior [76, 4]. Also, phase change memory [103, 109] is being used in neuron designs effectively.

2.3 Related Work on Neuromorphic System Design

Neuromorphic system design has been a very lucrative field in system design research industries. Because of the popularity of artificial neural networks and spiking neural networks, demand has emerged for hardware dedicated to neural network architecture.

Moreover, with the rise of neuromorphic computing, different research groups were eager to build some hardware implementations of neuromorphic systems. There have been works in digital, analog and mixed-signal design to build neuromorphic system. If we compare the architectures available, all have their own advantages and some draw-backs. For instance, the digital systems are often synchronous and more robust but they are also more power hungry whereas the analog systems are typically asynchronous and energy efficient. But the analog systems are a bit noisy and less prone to probabilistic noises. To implement a fully digital neuromorphic system, FPGAs are useful as they have a programmable fabric easily programmable for any working system. For instance, [15] presents an FPGA implementation of a neuromorphic system where one million neurons have been included. Neurons were defined as arithmetic logic units and a fully digital approach has been used to implement the system but a full neuromorphic system was not realized. More specifically, the hardware was not fully capable of doing extensive computation which a neuromorphic hardware can achieve. So, IBM came up with a fully custom ASIC neuromorphic chip named TrueNorth [38] fabricated using Samsung's 28nm process. The system contains 256 million synapses with over 1 million neurons. TrueNorth is a synchronous deterministic neuromorphic system and it is being used to execute several neuromorphic applications. Another example of an ASIC neuromorphic system is SpiNNaker [33] by the University of Manchester research group. The system contains ARM processors, local and shared memory, and peripherals for general system support. Since they use a conventional processor, the processing unit is not customized for neuromorphic activities but the integration and connection of several SpiNNaker chips gives the flexibility to build a larger system. Both TruNorth and SpiNNaker are designed as spiking neural network architectures with reported energy consumption in the pJ-nJ range. There are other similar hardware projects such as BrainScaleS [82], Neurogrid [32] etc.

Apart from digital ASIC designs, several other analog and mixed-signal approaches have also been explored as well. For instance, Carver Mead [62] introduced neuromorphic computing as an analog VLSI implementation where all the synapses and neurons were presented with pure analog implementation. Then there is the famous silicon retina [61] where a thin sheet of retina is built using analog sensors. Moreover, there are several

familiar characteristics of biological signals and analog components that make the retina suitable to implement analog neuromorphic systems. Another approach introduced by Mead is the sub-threshold mode of operation for analog neuromorphic implementations. Later research led to analog neuromorphic hardware [8, 21] based on the power efficiency argument because running sub-threshold would help in reaching drastically improved energy efficiency. However, the sub-threshold operation again can slow down the total system.

Considering the von Neumann bottleneck with increasing demand of neuromorphic architectures, researchers have also explored hybrid systems that include a CMOS process and several new emerging devices. The memristor is one such promising device and has been used in building neuromorphic systems where area density and low energy have been driving forces. Initially, researchers proposed a nano-molecular device acting as an active synapse in the presence of CMOS neurons [56, 97]. So, the advancement in technology made it possible to place both CMOS and non-silicon devices together in a chip. Also, the crossbar architecture of the nano-material/memristors have been proposed because of the area density. Later, many researchers began working with memristor modeling and playing with different memristor materials and models. Hence, several researchers are now considering hybrid memritive-CMOS neuromorphic systems like [41, 105, 90]. Hopefully, the addition of interesting research each and every day will lead this platform to a level where the neuromorphic system could help in accelerating the computing power for the next generation.

2.4 Background on Proposed Neuromorphic System

The proposed works is on designing a CMOS memristive neuromorphic system. The idea of this architecture is inspired from the work by Schuman *et. al* [85]. In [85], a neuroscience-inspired dynamic architecture or NIDA is introduced which is a 3D spiking neural network architecture (shown in Fig. 2.1). This architecture includes neurons and synapses as computing elements in 3D space. This way, it can contain the information including time and delay and hence compatible for dynamic network such as recurrent neural network (RNN) architecture. An RNN is capable of storing information for the previous cycles and later

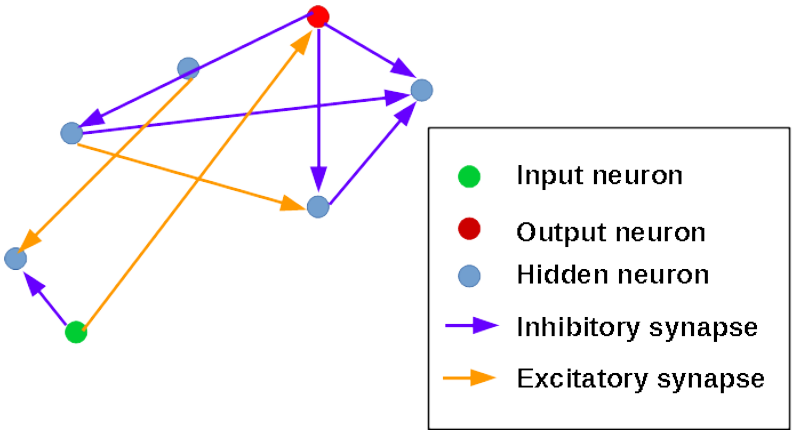

Figure 2.1: An example of NIDA network with different varieties of neurons and synapses.

helping in future computation providing those information. Since NIDA contains features of a continuous RNN architecture such as storing synaptic delay and spiky event generation, NIDA is useful in analyzing spatio-temporal data.

The NIDA networks are generated using a genetic algorithm called Evolutionary Optimization (EO) and the networks contain both neurons and synapses. Being a 3D spiking architecture, NIDA neurons and synapses are both spaced in space. Synapses are of two types: inhibitory and excitatory. The synapses in NIDA are defined by their connection to the corresponding neurons and store synaptic weights to regulate charge accumulation. They also represent synaptic delay as a part of dynamic behavior. The neurons are the computational nodes that generate firing event and hence, NIDA has a spiking network architecture. The neurons also contain information about threshold and refractory period. The interesting feature of the NIDA is that it generated very small and sparse networks that are recurrent in nature. Thus, this architecture is more useful in solving neural network problems with smaller but more efficient networks than conventional deep learning architecture.

Since NIDA is built on high-level simulation, a hardware implementation proved its efficiency in connectivity and recurrent features. This hardware implementation is named as dynamic adaptive neural network array or DANNA [25]. This is an FPGA implementation of NIDA which is also dynamic in nature and compatible with RNN features. DANNA contains neurons and synapses as computing elements. Each element can be represented as either

synapse or neuron and each are connected to its neighboring elements. This architecture is also event based and works well with spatio-temporal networks.

The implementation of DANNA inspired to work more on designing a system which is more area and energy efficient because DANNA is implemented on FPGA and requires a considerable amount of area and power. This need of reduction in area and energy consumption led to the design of a CMOS memristive neuromorphic system which is named as mrDANNA. This is a fully custom CMOS implementation. Though NIDA is asynchronous in nature, mrDANNA is a synchronous implementation of NIDA and works on a digital system clock. The main inspiration behind this work is building a system that contains the dynamic feature of NIDA (suitable for RNN) while being energy and area efficient both in circuit and system level.

Chapter 3

Twin Memristive Synapse and Mixed-Signal Neurons

3.1 Memristive Synapse Design

In biological neuronal systems, synaptic components play a vital role in transferring signals from one node to another. Since neuromorphic synapses are inspired from biological synapse, they are a major component of any neuromorphic system design. According to the existing works on synapse design inspired from the biological brain, a synapse can be constructed in two major ways; one can be defined as a spiking based synapse and the other is event based synapse. Both types contribute in synapse architecture based on the necessity of the specific neuromorphic system and there are several works on designing synapse circuitry based on these approaches. Initially, most works in designing synapse circuits involve fully CMOS implementation since the technology is well established for semiconductor devices. Unfortunately, the CMOS synapse implementation is facing the von Neumann bottleneck of sizing. Consequently, energy and area issues are becoming more prominent with the advancement of technology. Hence, researchers have begun to explore other materials and devices for designing synapses that help reduce the area. People have considered several two and three terminal devices such as phase-change memory [29, 98, 52, 107], ferroelectric devices [71, 106], floating gate transistors [79, 117] and memristors [2, 108] while designing synapses for neuromorphic system. Among these implementations, memristive synapses are

non-volatile and multi-resistive, particularly promising characteristics for artificial synapses. Moreover, memristive are good solutions for implementing area and energy efficient synapse structure.

3.1.1 Background of Memristor

Memristors are one of the four basic circuit components. It was first theorized by Leon O. Chua [22] in 1971, representing the missing link between the electric flux and charge. The term Memristor, is a conjunction of "memory resistor" as they are two terminal nanoscale devices that exhibit switching resistance and non-volatile in nature. One interesting feature of memristors is that its resistance can be modulated by changing the voltage applied across the device. A memristor has two extreme resistance limits called low resistance state (LRS) and high resistance state (HRS). The device will switch from one state to another when a switching voltage is applied for a certain amount of time across it. Moreover, it can attain any resistance level based on the magnitude of the voltage applied and the amount of time the voltage is applied. Hence, memristors have the characteristics of storing different resistance levels, which is analogous to artificial synapses in spiking neural networks. While the memristor is switching from one resistance state to another, the minimum amount of voltage applied for switching is called the threshold voltage and the minimum amount of time required is the switching time. Threshold voltages could be different for (HRS to LRS) and (LRS to HRS) switching and are referred to as positive threshold voltage (V_{tp}), and negative threshold voltage (V_{tn}). Similarly, the switching time is also different for (HRS to LRS) and (LRS to HRS) switching and are referred to as positive switching time (t_{swp}), and negative switching time (t_{swn}), respectively. There are several materials that show the characteristics of memristors including TaO_x [114], TiO_2[64], HfO_x [53], chalcogenides [54, 73], silicon [13, 66], organic materials [10], ferroelectric materials [20, 75], carbon nanotubes [45], etc. Each memristive material is differentiated by its LRS values, LRS to HRS ratios, threshold voltages, and switching times. A good range of LRS and HRS values (Table 3.1) has been considered for the proposed memristive synapse design based on the available materials presented in the literature.

Table 3.1: Switching parameters for metal-oxide memristors [17]

Parameter \ Devices (mean)	TaO_x [114]	HfO_x [104]	TiO_x [65]	Parameter variance
HRS	$10k\Omega$	$300k\Omega$	$2M\Omega$	$\pm 20\%$
LRS	$2k\Omega$	$30k\Omega$	$500k\Omega$	$\pm 10\%$
V_{tp}	0.5V	0.7V	0.5V	$\pm 10\%$
V_{tn}	-0.5V	-1.0V	-0.5V	$\pm 10\%$
t_{swp}	105ps	10ns	10ns	$\pm 5\%$
t_{swn}	120ps	1μs	10ns	$\pm 5\%$

Here, the memristor model used for simulation is derived from a model previously developed in [7]. Our model specifically emphasizes the bipolar behavior considered in previous related works [104]. While performing a SET operation from HRS to LRS, the resistance change in the memristor is given by:

$$R_{new} = R_{initial} - \frac{\Delta r \times |V(t)| \times t_{pw}}{t_{swp} \times V_{tp}}. \tag{3.1}$$

The resistance change during the RESET operation is given by:

$$R_{new} = R_{initial} + \frac{\Delta r \times |V(t)| \times t_{pw}}{t_{swn} \times V_{tn}}, \tag{3.2}$$

where R is the resistance of the memristor, Δr is the absolute difference between the HRS and LRS values, $V(t)$ is the applied voltage across the memristor and t_{pw} is the time duration for an applied voltage pulse. Assuming the memristors have symmetric switching time and threshold voltage, the change in memristance (ΔR) in either direction is given by:

$$\begin{aligned} \Delta R &= R_{new} - R_{initial} \\ &= \frac{\Delta r \times |V(t)| \times t_{pw}}{t_{sw} \times V_{th}}, \end{aligned} \tag{3.3}$$

where $t_{sw} = t_{swp} = t_{swn}$ and $V_{th} = V_{tp} = V_{tn}$. An example current-voltage relationship of the memristor model used in this work is shown in Fig. 3.1,

Memristors being non-volatile and programmable by nature make them a good fit for designing artificial synapses because they can achieve variable resistance states which refers

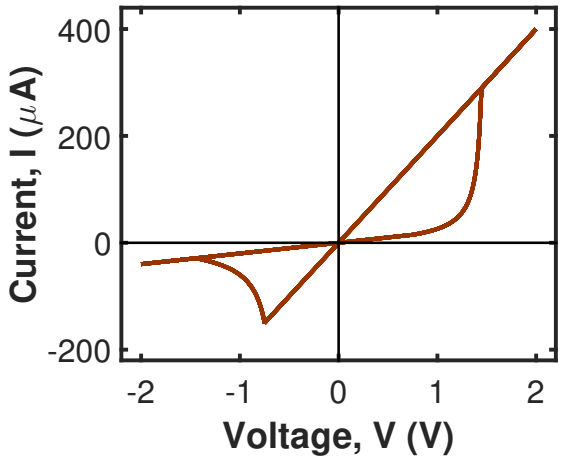

Figure 3.1: Memristor current-voltage relationship.

to different synaptic weights. As synapses, memristors are able to transmit weighted inputs to the connected neurons. The neuron then leverages the analog current output of the memristive synapse to generate firing events or spikes that are digital and synchronous to the system. Moreover, the system considered here follows an unsupervised Long Term Plasticity (LTP) mechanism for online learning. This learning method enables the dynamic synaptic weight adaptability based on the temporal relationship of the pre- and the post-synaptic fires which is driven by the pre-neuron connection to the LTP control block and the necessary feedback signal from the post-synaptic neuron.

3.1.2 Synapse Structure

The synapse structure considered in this design (shown in Fig. 3.2) consists of two memristors connected back to back, referred to as a twin memristive synapse. The synaptic weight is stored using the pair of memristors where the input voltages across the memristive weights yield a weighted sum in the form of a current. Basically, the idea here is that the current flowing through the synaptic node is proportional to its weight and hence depends on the resistances of the two memristors. This approach of using weighted current to represent synaptic weight is very similar to several other memristor-based neural network designs available in the literature [80, 67, 40, 48].

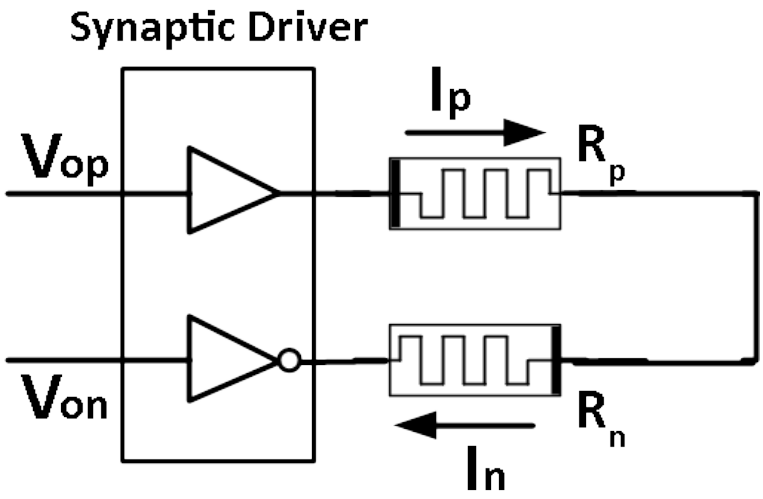

Figure 3.2: Twin memristor synapse architecture along with its synaptic driver.

Ideally, a single memristor can represent a single weight. To represent both positive and negative weights a minimum of two memristors are required in a synapse design. Since we are building recurrent spiking neural networks being inspired from biological phenomena, we have considered the inhibitory and exhibitory connection from one neuron to another. Here, exhibitory connections are based on the positive weight whereas the inhibitory one follows from the negative weight [77]. There have been several approaches proposed in the literature for implementing dual weights. For instance, ideas have been explored [40, 101, 102, 116] that represent negative components of the weights using a twin memristive crossbar. The idea behind using the twin crossbar is to represent each weight with a separate crossbar. If M^+ crossbar represents positive weight, there will be a M^- crossbar with inverse weights for the negative weight. In fact, in [101], research showed that identical crossbars can be used instead of inverse crossbars for representing both the weights. In both cases the twin crossbar architecture is considered. Moreover, there are other works with memristive crossbars [39, 5, 47, 50, 112] to mimic human brain. All of these works using crossbars consider some area overhead for controlling and programming circuits and are not prone to sneak-path currents. On the other hand, the twin memristive configuration is smaller in size compared to crossbars and peripherals and specifically considered to build synapses with positive and negative weight features for simple neuromorphic system core. In addition, our

goal is to design synapses for SNNs which are very sparse and don't need fully connected dense layers like deep neural networks and hence the design is area efficient.

In the twin memristive synapse shown in Fig. 3.3, each memristor drives current in a single direction with the memristors are connected in opposite directions as mentioned earlier. Thus, one memristor is responsible for driving a positive current while the other memristor pulls the current or drives a negative current. For the twin memristive synapse, one terminal is connected to respective input voltages whereas the common terminal connects to a post-synaptic mid-rail voltage. The mid-rail voltage can be defined as the median voltage of the high and low rail voltages. Depending on the design setup, this voltage is connected as virtual ground since this node is actually an input port of an integrator op-amp which will be discussed in detail in section 3.2. So, the twin memristive synapse connected to separate voltages produces an effective current which depends on the relative values of the resistances in the twin memristive synapse. Then the synaptic weight is proportional to the effective current alongside the effective conductivity of the twin memristive pair shown in equation 3.4.

$$G_{eff,i} \propto W_i \tag{3.4}$$

Here, $G_{eff,i}$ is the effective conductance of the i^{th} synapse and W_i is its synaptic weight.

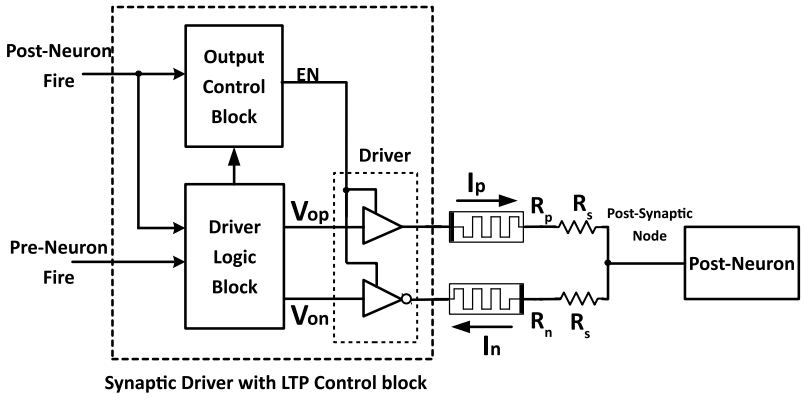

Figure 3.3: Twin memristor synapse along with its control block providing the interlink between the pre- and post-neuron [17].

This equation shows that there is a linear relationship between the effective conductivity of any twin memristor and the weight of the corresponding synapse. To model the synaptic weights based on the memristors, the following relations are defined.

$$G_{eff,i} = W_i.G_{eff,1} \tag{3.5}$$

$$\begin{aligned} W_i.G_{eff,1} &= \frac{1}{R_{p,i}} - \frac{1}{R_{n,i}} \\ &= \frac{1}{R_{p,i}} - \frac{1}{LRS + HRS - R_{p,i}}; \\ where R_n &= LRS + HRS - R_p \end{aligned} \tag{3.6}$$

A twin memristive synapse has a limitation in synaptic weight mapping based on the values of HRS and LRS which in turn controls the effective conductance of the synapse. So, different effective conductance can be achieved by different combinations of R_p (resistance of memristor in positive direction) and R_n (resistance of memristor in negative direction) according to equation 3.6. When R_p is equal to LRS and R_n is same as HRS, the maximum effective conductance (G_{max}) can be achieved. On the contrary, when both R_p ans R_n are equal, the effective conductance would be minimum for that synapse and it would represent synaptic weight of "0". For instance, when both values of R_n and R_p are equal to the average allowed resistance $(HRS + LRS)/2$, a synaptic weight of "0" is achieved. Initially, it is assumed that the synaptic weight change is approximately symmetric in both directions from the median of LRS and HRS meaning that the change ΔR_n=ΔR_p. Hence the values of initial R_n and R_p need to be initialized at an equal distance from the median of HRS and LRS assuming R_n+R_p=LRS+HRS. The synaptic weight as well as the resistance in the memristors change after initialization based on the values of ΔR_n and ΔR_p as a result of online learning. So, the resistance of the memristors for each synaptic weight can be represented in the following way.

$$\begin{aligned} R_{p,i} = \frac{HRS + LRS}{2} + \frac{1}{W_i.G_{eff,1}} \\ + \frac{1}{2}.\sqrt{\left[(HRS + LRS)^2 + \frac{1}{(W_i.G_{eff,1})^2}\right]}. \end{aligned} \tag{3.7}$$

It is to be noted that the currents through the twin memrisotrs would be similar if the resistance values are equal in the memristive pair and that way the currents would cancel each other resulting in a synaptic weight of zero. Similarly, if R_p is lesser (greater) than R_n, the weight is positive (negative). In the synapse design, there is a driver logic block which supplies driving voltages to the memristor pair to keep the synapse operating in either of its two phases of operation and those are *accumulation* and *learning*. The synapse is in *accumulation* phase when there exists a pre-neuron firing event. During this phase the synaptic control block provides the driving force to make a positive current flow through R_p and a negative current through R_n. It is to be noted that during the *accumulation* phase, the post-synaptic node is ensured to rest on the mid-rail voltage by forcing the input node of the post-neuron to virtual ground. When the synapse is in *learning* phase there exists post-neuron firing events. During this phase two opposite phenomenon could occur on the synaptic weight update. If the pre-neuron fire arrives just before the post-neuron fire, the corresponding synapse weight would be potentiated or increased. On the other hand, if the pre-neuron arrives just after the post-neuron, the synapse weight would be depressed or decreased. This dynamic synapse weight update follows the famous STDP rule which is inspired from the learning in the biological neural networks.

3.1.3 Digital Long Term Plasticity (DLTP)

According to the existing literature, most neural networks are trained using popular learning algorithms, for example back-propagation or supervised gradient descent learning. These learning algorithms are mostly offline learning topologies that help the neural networks train well using an available dataset. However, these are inefficient for online learning which is a prominent feature in biologically inspired spiking neural networks. An online learning mechanism is necessary to make the networks learn online or during run-time. Long Term Plasticity (LTP) is one of the widely used online learning mechanisms which helps the network as well as the circuit learn online by continuously updating the synaptic weights based on the pre- and post-neuron fires timing. Several works have been developed where the circuits are trying to mimic synaptic plasticity behavior [111]. STDP technique is popular in modeling LTP. The most interesting and mostly used techniques used by the

prior works include modifying the magnitude of the applied voltages across the synapses. This is achieved by taking account the time difference between the pre and post neuron fires and an applied voltage tail that creates the variation in applied membrane voltage to update synaptic weights.

Unlike other prior works, this design has a different approach in implementing online learning for the system. Since the total system is mixed-signal in nature, the feature is leveraged to develop circuits for online learning. Instead of crafting analog voltage tails precisely, a digital pulse modulation method has been utilized here to implement a digital LTP (DLTP). This DLTP process is based on tracking the timing of pre- and post-neuron fires based on clock cycles. This algorithm refers to a single clock cycle only, meaning if there is a post-neuron fire, the DLTP circuit considers pre-neuron fires in the cycles right before and after the post-neuron fire. If there is any pre-neuron fire present before the post-neuron fire, the synapse weight is increased or potentiated. On the other hand, if it arrives after the post-neuron fire, the synaptic weight will be decreased or depressed. Since DLTP considers the weight update based on a single clock cycle tracking, it can be referred to as the one clock cycle tracking version of STDP which is a famous learning implementation introduced in different learning circuits [95, 11, 46, 91]. Being a single cycle tracking version of STDP, DLTP has several advantages over STDP. For example, implementing a detailed STDP learning rule for several clock cycles would result in area overhead and hence more energy whereas, DLTP acts similarly but with lower area and energy consumption.

The effective conductance of the twin memristor shown in Fig. 3.3 can be defined by the following equation:

$$G_{eff} = \frac{1}{R_p} - \frac{1}{R_n} \tag{3.8}$$

If there is any synaptic weight update because of DLTP, the weight change in the resistance values of the twin memristors ΔR for both potentiation and depression are assumed to be the same. Considering a potentiation scenario, the new effective conductance can be defined

by:

$$
\begin{aligned}
G_{eff,pot} &= \frac{1}{R_p - \Delta R} - \frac{1}{R_n + \Delta R} \\
&= \frac{1}{R_p\Big(1 - \frac{\Delta R}{R_p}\Big)} - \frac{1}{R_n\Big(1 + \frac{\Delta R}{R_n}\Big)} \\
&= \frac{1}{R_p}\Big(1 - \frac{\Delta R}{R_p}\Big)^{-1} + \frac{1}{R_n}\Big(1 + \frac{\Delta R}{R_n}\Big)^{-1} \\
&= \frac{1}{R_p}\Big[1 + \frac{\Delta R}{R_p} + \Big(\frac{\Delta R}{R_p}\Big)^2 + ...\Big] - \frac{1}{R_n}\Big[1 \\
&- \frac{\Delta R}{R_n} + \Big(\frac{\Delta R}{R_n}\Big)^2 - ....\Big] \\
&= \frac{1}{R_p} - \frac{1}{R_n} + \Delta R\left(\frac{1}{R_p^2} + \frac{1}{R_n^2}\right) + \Delta R^2\Big(\frac{1}{R_p^3} \\
&- \frac{1}{R_n^3}\Big) + ... \\
&= G_{eff} + \Delta R\,(G_p^2 + G_n^2) + \Delta R^2\,(G_p^3 - G_n^3) + ...
\end{aligned}
\tag{3.9}
$$

Thus, the change in the effective conductance can be described by:

$$
\begin{aligned}
\Delta G_{pot} &= G_{eff,pot} - G_{eff} \\
&= \Delta R\,(G_p^2 + G_n^2) + \Delta R^2\,(G_p^3 - G_n^3) + ...,
\end{aligned}
\tag{3.10}
$$

and for positive weights ($R_p < R_n$) the change would be higher than that of the negative weights ($R_p > R_n$).

Next we consider the reduction in weight and the new depressed effective conductance will be:

$$
\begin{aligned}
G_{eff,dep} &= \frac{1}{R_p + \Delta R} - \frac{1}{R_n - \Delta R} \\
&= \frac{1}{R_p\Big(1 + \frac{\Delta R}{R_p}\Big)} - \frac{1}{R_n\Big(1 - \frac{\Delta R}{R_n}\Big)} \\
&= \frac{1}{R_p}\Big(1 + \frac{\Delta R}{R_p}\Big)^{-1} + \frac{1}{R_n}\Big(1 - \frac{\Delta R}{R_n}\Big)^{-1} \\
&= G_{eff} - \Delta R\,(G_p^2 + G_n^2) + \Delta R^2\,(G_p^3 - G_n^3) - ....
\end{aligned}
\tag{3.11}
$$

Thus, the synaptic weight change which is proportional to the effective conductance change would be:

$$\Delta G = -\left[\Delta R\left(G_p^2 + G_n^2\right) - \Delta R^2\left(G_p^3 - G_n^3\right) + ...\right], \tag{3.12}$$

and similarly we can say that the change would not be perfectly equal for both positive and negative weights. It is to be noted that the memristor device parameters and choice of clock frequency ensures ΔR to be smaller than both R_p and R_n. Hence the binomial series expansion is valid for both cases.

The circuit level implementation of DLTP consists of two important blocks. One is the output control block that generates an enable signal to switch on the potentiation/depression and the other one is the driver logic block. The output control block generates an enable signal sensing the firing of post-neurons caused by any pre-neuron fires following

$$EN = \overline{\overline{F_{post} * F_{pre\_t}} * F_{pre\_b}}, \tag{3.13}$$

where F_{post} is the signal from the post-neuron, F_{pre_t} is a delayed signal from pre-neuron and F_{pre_b} is the inversion of the pre-neuron signal. The EN signal is also asserted during the *accumulation* phase so that V_{op} and V_{on} can drive positive and negative currents through R_p and R_n, respectively.

The synapse driver logic block (shown in Fig. 3.4) generates both the positive (V_{op}) and negative (V_{on}) driving voltages to the memristors. During accumulation, R_p and R_n are driven to the positive and negative rails respectively. This is achieved by making $V_{op} = V_{on} =$

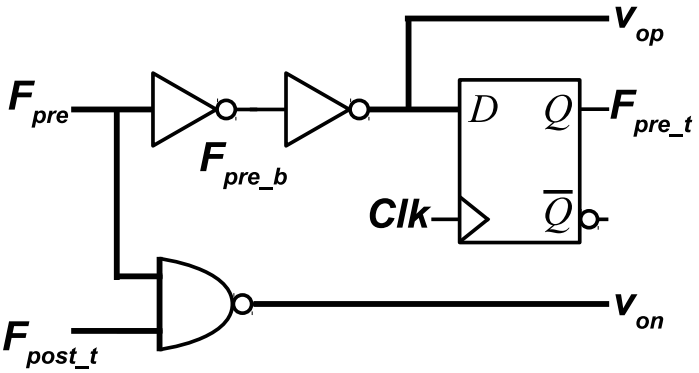

Figure 3.4: Driver logic block.

V_{DD}. It should be noted that the signal V_{on} drives an inverter to supply negative voltage (V_{SS}) on R_n (Fig. 3.3). Additionally, the post-synaptic node is held at virtual ground (mid-rail) so that the voltages across the memristors stay below the switching threshold of the memristor. This operational block is also responsible for supplying correct driving voltage to the twin memristor during the learning phase. If the control block senses a potentiation event, the driver logic block will operate in such a way that the voltage across the memristors R_p and R_n crosses the positive and negative thresholds, respectively, and hence the synaptic weight will increase following equation 3.10. So, for potentiation, $V_{op} = V_{SS}$ and $V_{on} = V_{DD}$ while the post-synaptic node is held at V_{DD} by the feedback from the neuron which will be described in section 3.2. This results in a rail-to-rail voltage drop across R_p and R_n. Since they are connected in opposite polarity, the value of R_p decreases while R_n increases, making the G_{eff} rise according to equation 3.8. Similarly, the depression logic is also dependent on the proper voltage across the memristors R_p and R_n crossing the threshold in the opposite direction. However, the post-synaptic node is also responsible for controlling DLTP.

A small network of synapses with two pre-neurons and a post-neuron is considered here to analyze the implementation of our DLTP approach. Fig. 3.5 shows the network with synapses containing weights of "1", two pre-neurons and a single post-neuron with a threshold of "2". The pre-neurons sends the synaptic signals to the corresponding synapses and the post-neuron receives that weighted signal. The post-neuron also generates post-synaptic fires which will be input to the next layer of pre-neurons. The pre-neuron inputs are digital pulse-

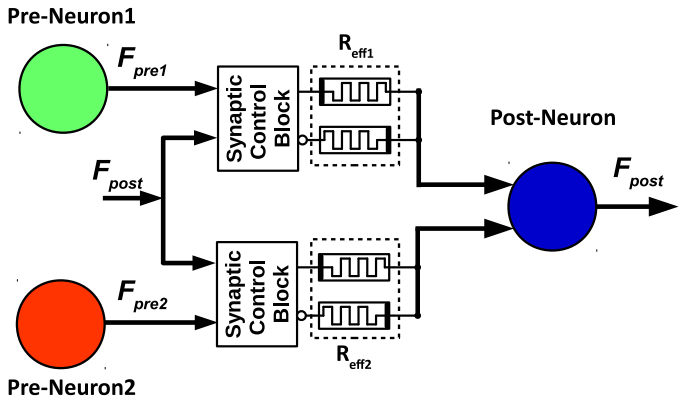

Figure 3.5: A single neuron connected with two synapses network presenting DLTP [17].

trains F_{pre1} and F_{pre2} and the post-neuron output is denoted by F_{post} (shown in Fig. 3.6). Since the DLTP circuit tracks the pre- and post-neuron spikes for a clock cycle before and after firing events, we can assume from the figure that both of the synapses will go through potentiation and depression in different clock cycles. In Fig. 3.6, G_{eff1} and G_{eff2} are the effective conductance values of the two synapses, primarily at an initial state based on the initial resistance of the memristive synapses. If we analyze the pre- and post-neuron spikes, we would see that the first post-neuron fire occurs after accumulating the charge of the first two F_{pre1} fires. So, the synapse R_{n1} is being potentiated and hence G_{eff1} is increased. On the other hand, the first fire of F_{pre2} is arriving simultaneously with post-neuron fire and it is not responsible for post-neuron fire. So, the synapse R_{n2} is being depressed and hence the effective conductance, G_{eff2} is decreased. However, the synaptic weight change will not be the same for each stage because with online learning the next weight change will be based on the updated weights.

3.1.4 Layout of the Synapse Circuit

The synapse layout was done in CMOS 65nm CMOS technology provided by SUNY Poly in order to fabricate the design. The design for the synapse includes a twin memristor connection, driver logic block and an output control block. These blocks are described in detail in section 3.1.3. The synapse layout includes the layout of this synaptic driver block as shown in Fig. 3.7 with the highlighted section in the figure showing one of the

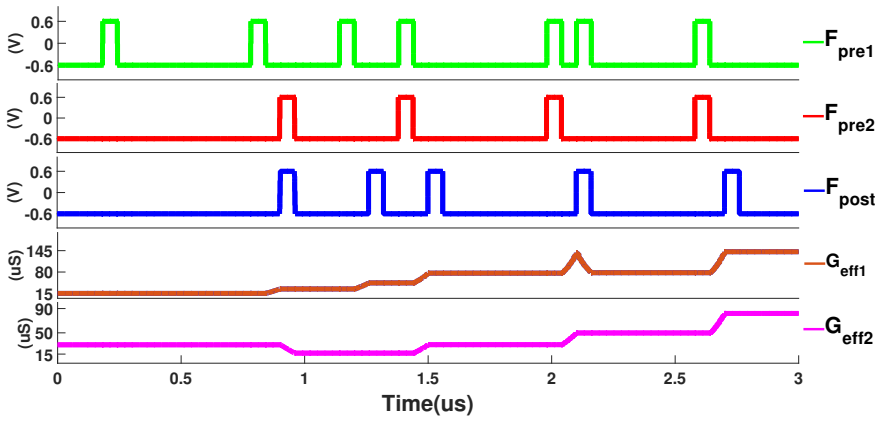

Figure 3.6: Simulation result for small DLTP network [17].

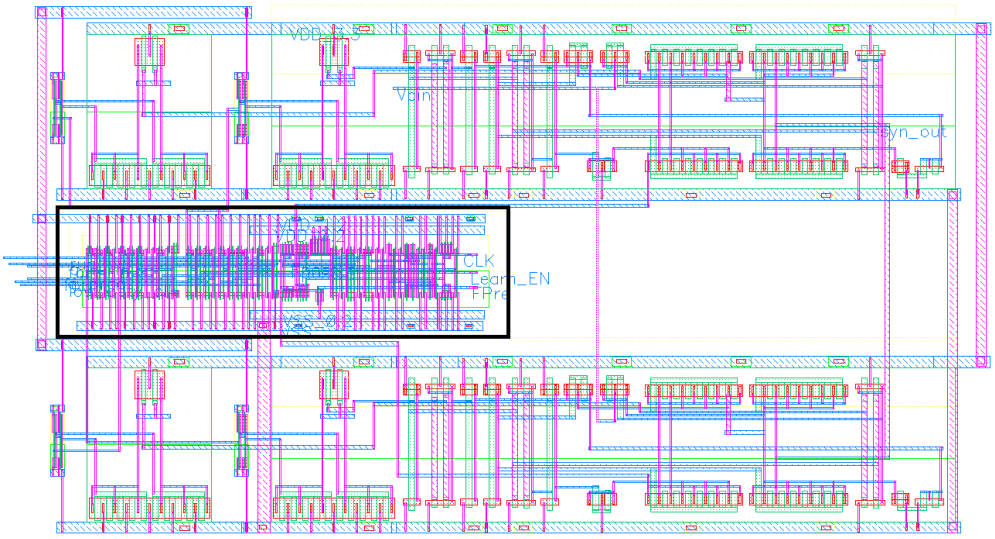

Figure 3.7: Layout of the memristive synapse with synaptic control circuits using 65nm technology node.

updated versions of the synaptic buffer (driver and the output control). In addition, there are other digital circuit components surrounding the synaptic buffer and the memristors. The memristors are typically laid out as an intermediate material between metal 1 and metal 2. The synapse layout also includes different NMOS for initializing memristors.

3.2 Mixed-Signal Neuron Design

The neuron is the component of a neural network where the weighted inputs from the synapses are summed together to generate an output signal or spike. Neurons can be biologically plausible or biologically inspired. Since we are more interested in spiking neural network (SNN), spike based neurons are specifically considered here. Hardware implementations of neurons become more prominent when the need for emulating a SNN needs to be more efficient. With increasing network size, the hardware needs to be specialized to take part in the neural computation. Thus, an efficient hardware implementation for the neurons is necessary.

Different approaches for implementing neurons in electronic hardware have been presented in the literature [43, 63, 70]. Depending on how these neurons are modeled the circuit level implementations vary. Some neurons are more inclined to neuroscience and are modeled

to represent the dendritic activities from the cell-level whereas some neurons are modeled to capture enough complexity to represent the functionality of biological neurons. Integrate and fire neurons are one of the more popular neuron architectures that covers the complexity level from the very basic integrate and fire approach to more complex computational models [44]. Here, an integrate and fire neuron is proposed which can be related to the state of the art considering it's robustness with integrating it into a large system.

The mechanism for the integrate and fire neuron is dependent on the membrane potential of the neuron. The inputs to any neuron is the weighted sum of pre-synaptic inputs that come from the synapses connected to the particular neuron. If the accumulated weighted sum is larger than a specified threshold, the neuron fires or it generates an output spike. Thus, the membrane potential V_{mem} can be defined in a following way.

$$C_{mem}\frac{dv(t)}{dt} = I_{in} + I_{leak} \tag{3.14}$$

Here, C_{mem}, I_{in}, I_{leak} and $v(t)$ are the membrane capacitance, input current from synapse to the neuron, leakage current through the membrane and the membrane voltage, respectively.

3.2.1 Neuron Functionality

The general behavior of an IAF neuron is to integrate charges from the incoming pre-synaptic inputs and generate a post-synaptic output as a fire event when the accumulated charge is higher than the threshold. The approach followed in this work leverages the advantage of mixed-signal design where the integration is done using the weighted input currents rather than voltage spikes and producing fire outputs as binary voltage pulse. This process is analog in nature while integrating but is digital for output spike generation. The IAF neuron designed for this work is presented in Fig. 3.8 which is similar to the design explained in [110].

The proposed neuron operates in two different phases. The first one is the *integration* phase and the second one is the $firing$ phase. When the neuron is in the integration phase, the op amp operates as an integrator. It accumulates charges from the incoming weighted current through the connected synapses. The feedback capacitor C_{fb} in Fig. 3.8 helps

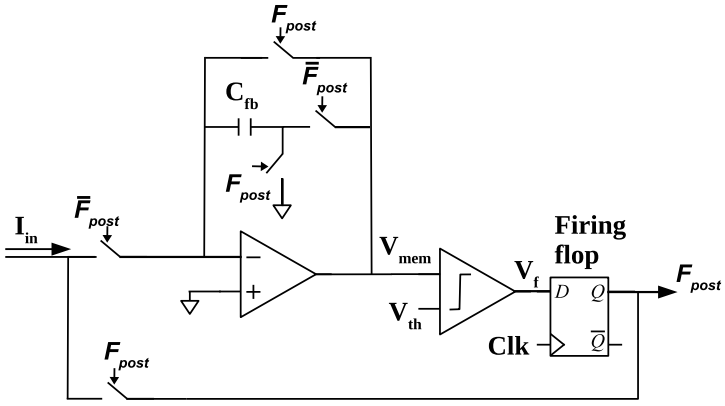

Figure 3.8: Mixed-signal Integrate and Fire (IAF) Neuron [17].

decide the accumulation rate for the integration. The accumulated charge is presented as a membrane potential V_{mem} and it keeps changing based on the accumulated charge. There is a comparator circuit that compares the membrane potential V_{mem} against a threshold voltage V_{th} and helps in generating output values. An output value is then used by the firing flop to generate a firing pulse that is synchronized to a system clock. One additional thing to note here is that the reference voltage, V_{ref} is tied to a mid rail voltage which is "0V" for this design. This reference voltage is leveraged to ensure that the "-" input of the op-amp remains in the virtual ground while the neuron is in accumulation phase.

During the firing phase, the op amps operates as a buffer which helps in resetting the active charge potential available as the membrane voltage. The firing phase also enables the feedback mechanism which in turn results in activating the online learning mechanism (DLTP) to make the synaptic weights adaptive to the results. Based on the DLTP mechanism described in 3.1.3, if any input synapse arrives just before to a firing event, it is correlated with the output fire and a potentiation occurs such that the synaptic weight of that particular synapse is increased. On the other hand, if the synaptic input arrives at the same time as the output firing event, depression occurs making weight of the synapse decrease. This ensures that the input node of the IAF neuron plays an important role in driving the single cycle online learning DLTP process. Also, while the neuron is in the firing phase, the feedback control circuit supplies a voltage potential to one side of the twin memristor synapses so that the weight is altered based on the voltage present across the memristors. In addition, the feedback mechanism helps in establishing an one cycle refractory period, meaning the

neuron would be idle for one clock cycle after it has gone through a firing event. When the neuron is in its refractory period, it will not accumulate charge for any incoming synaptic input current.

One additional feature of this neuron design is that the neuron is implemented to reset itself after a firing event. Then it starts preparing itself for the upcoming input spikes. This implementation also ensures that the neuron operating phases are synchronous with the system clock.

3.2.2 Neuron Components

Based on the characteristics of the IAF neuron, our neuron is composed of three main component blocks. These are the integrator, comparator and the digital control part. In Fig. 3.9, transistor M_0-M_{12} constructs the integrator part with the feedback capacitor C_{fb}. The integrator is basically a three stage op-amp designed to operate in a range of $20 - 25MHz$, consistent with other existing works [59]. Moreover, we are using twin memristive synapse with metal oxide materials and considering the switching time, we settled on a system frequency that can provide a good range of analog resistive values. The integrator takes in input current I_{in} and the bias voltage, while V_{bias} on the gates of M_4, M_9 and M_{12} secure the biasing current and the integration of the input current. The next stage of the neuron is the comparator which is comprised of another op-amp containing the M_{13}-M_{19} transistors in Fig. 3.9. This op-amp is a two stage amplifier with higher output resistance. The bias voltage works similar to the integrator part. The comparator takes in two input voltages.

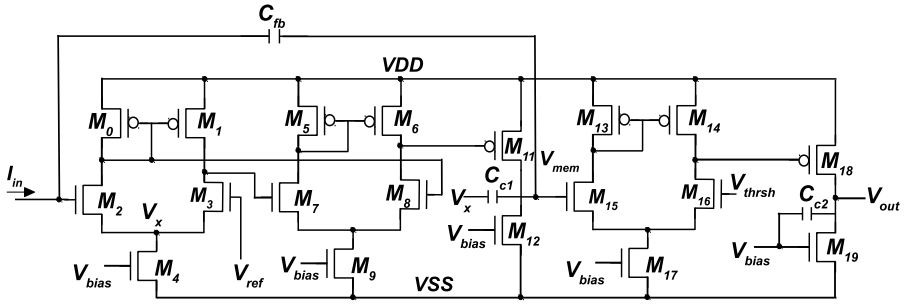

Figure 3.9: Analog integration of charges and comparison with neuron threshold [17].

One comes directly from the output of the integrator which is the membrane potential V_{mem} with the other being an external input which is the threshold. The comparator drives a high voltage output when the membrane potential is higher than the threshold.

Since the core of the neuron is analog in nature and the connection from the core to the system is digital, we would like to design the neuron in such a way that the output from the neurons can be fed to the post-synapses as digital inputs. Hence, there are some digital components such as flops and transmission gates that help generate digital pulses for each output spike. Moreover, there are additional digital circuits that help establish an one cycle refractory period mentioned in the earlier section so that the neuron gets enough time to reset itself to the resting mid-rail voltage before it starts integrating the next set of input signals.

3.2.3 Layout of the Neuron

The mixed-signal neuron layout was completed using Cadence Virtuoso tool with 65nm process from SUNY Poly. The layout is shown in Fig. 3.10. This layout has five different metal layers from M1-BB and it also includes the polysilicon, oxide and dopant layers. Two different capacitors are used in this layout. One is the internal feedback capacitor C_{c1} with a capacitance value of $357.76fF$ which is laid out on top. The other is the integrator feedback capacitor C_{fb} with a value of $670.814fF$. The C_{fb} capacitor is the larger one drawn on the bottom of the layout in Fig. 3.10. The total height of the neuron layout is $61.08\mu m$ and the width is $56.59\mu m$. The testing strategy for testing the neuron is included in appendix B.1.

3.3 Synapse and Neuron Test Structures

Synapses and neurons are the component blocks of the neuromorphic computing system. For this work, memristive synapses and mixed-signal neurons have been designed and verified in isolated design tests. Some tests also have been designed to verify the characteristics of each component and their behavior when connected together. Hence, some test structures have been considered with different synapse and neuron combinations and these test structures

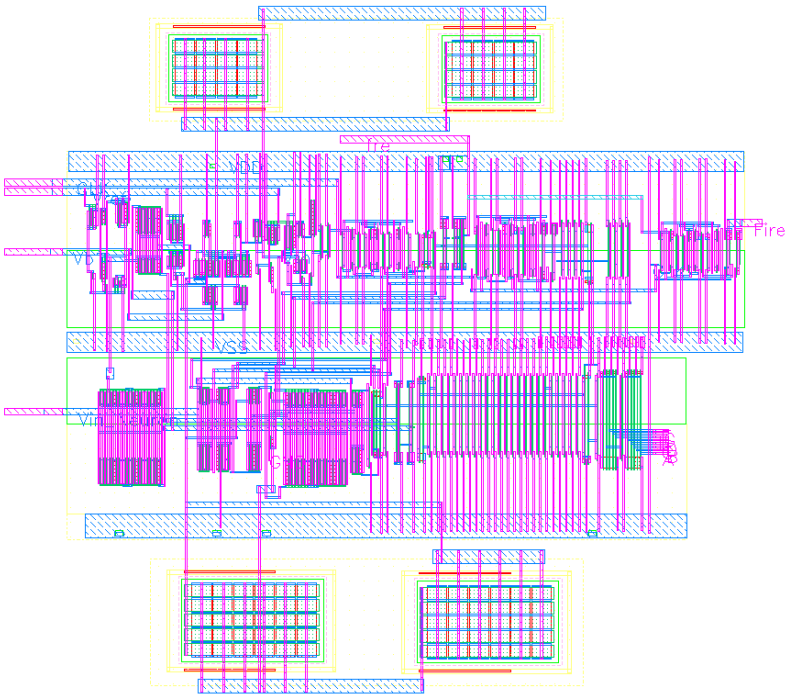

Figure 3.10: Layout of the mixed-signal neuron with 65nm technology node.

are also available in the fabricated die to test physically in near future. The testing strategies are included in appendix B

3.3.1 Single Resistive Synapse and a Mixed-Signal Neuron

This test structure includes a single resistor as a resistive synpase connected to the mixed-signal neuron described in section 3.2. The test is used to verify the neuron functionality when driven by a simple synapse. The neuron receives the weighted pre-synaptic input current based on applied "SFire" and accumulates charge until the accumulated voltage is higher than the threshold voltage V_{th}. Then the neuron generates a spiking voltage "Fire1" as an output. The schematic of this test structure is shown in Fig. 3.11 and the layout is also provided in Fig. 3.12. Details for the pin structures and assignments are provided in the appendix.

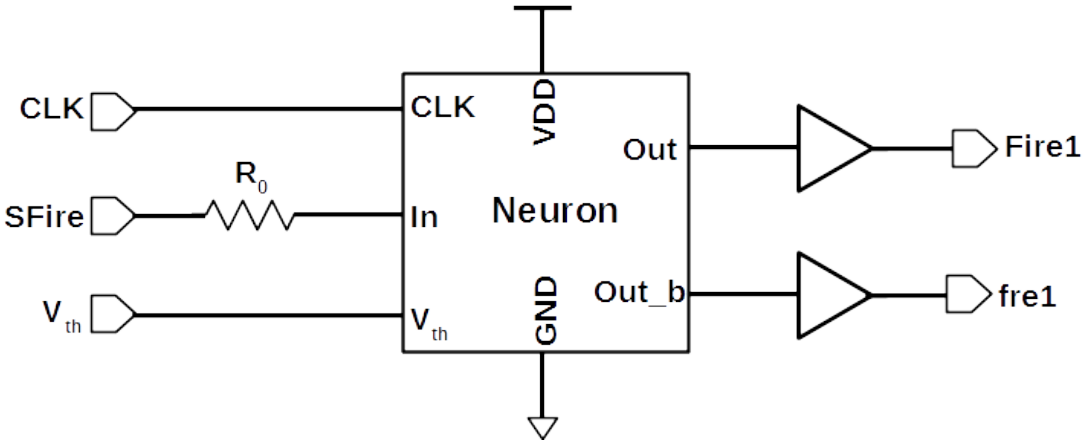

Figure 3.11: Schematic of the single resistor with mixed-signal neuron with 65nm technology node.

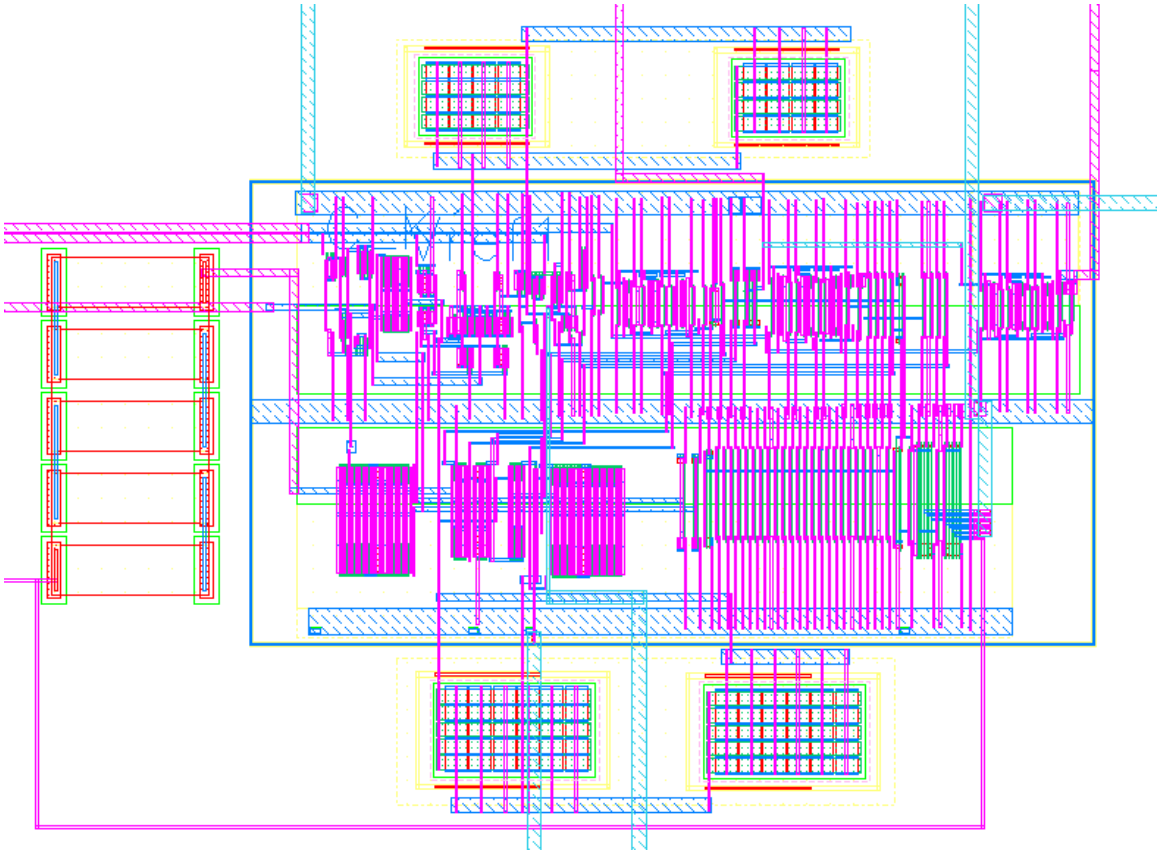

Figure 3.12: Layout of the single resistor with mixed-signal neuron with 65nm technology node.

3.3.2 Multiple Resistive Synapses and a Mixed-Signal Neuron

This test structure emulates the combination of two different synapses (resistors) driving a single mixed-signal neuron. Here, the resistor values are chosen in a way so that the conductance matches the highest synaptic weight possible for the memristive synapses considered. Since the equivalent resistance values represent high conductivity, this test structure is also a representation of a single synapse and a single neuron connection. Moreover, this test also verifies the accumulation and firing of a neuron like the previous test.

In this test structure, two inverted input signals are supplied to the resistors connected in parallel. The neuron has threshold voltage V_{th} and a CLK clock signal. The output of the neuron is denoted by $Fire$ signal. The schematic of this test structure is shown in Fig. 3.13 and the layout of the test structure is presented in Fig. 3.14. The detailed simulation result with pin assignments are provided in appendix B.1

3.3.3 Memristive Synapse with Forming and Programming Circuit and a Mixed-Signal Neuron

This test structure is important to verify that the forming of the memristors happen successfully because without forming the memristors are only regular resistors.

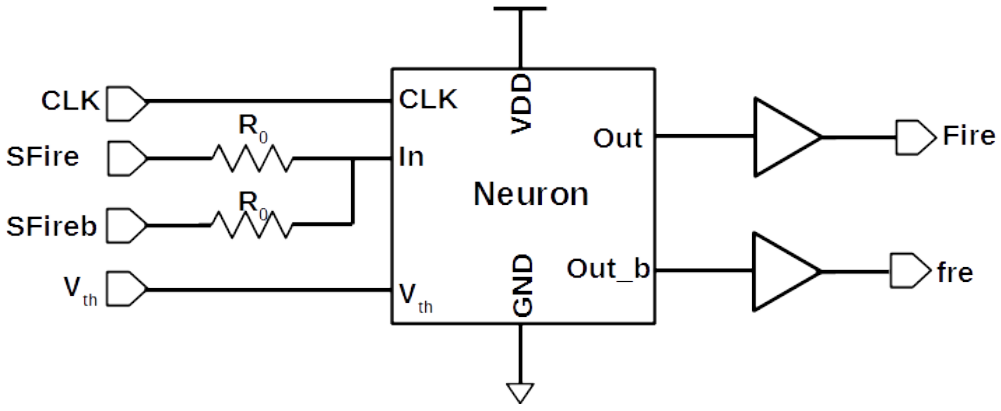

Figure 3.13: Schematic of the multiple resistors with mixed-signal neuron with 65nm technology node.

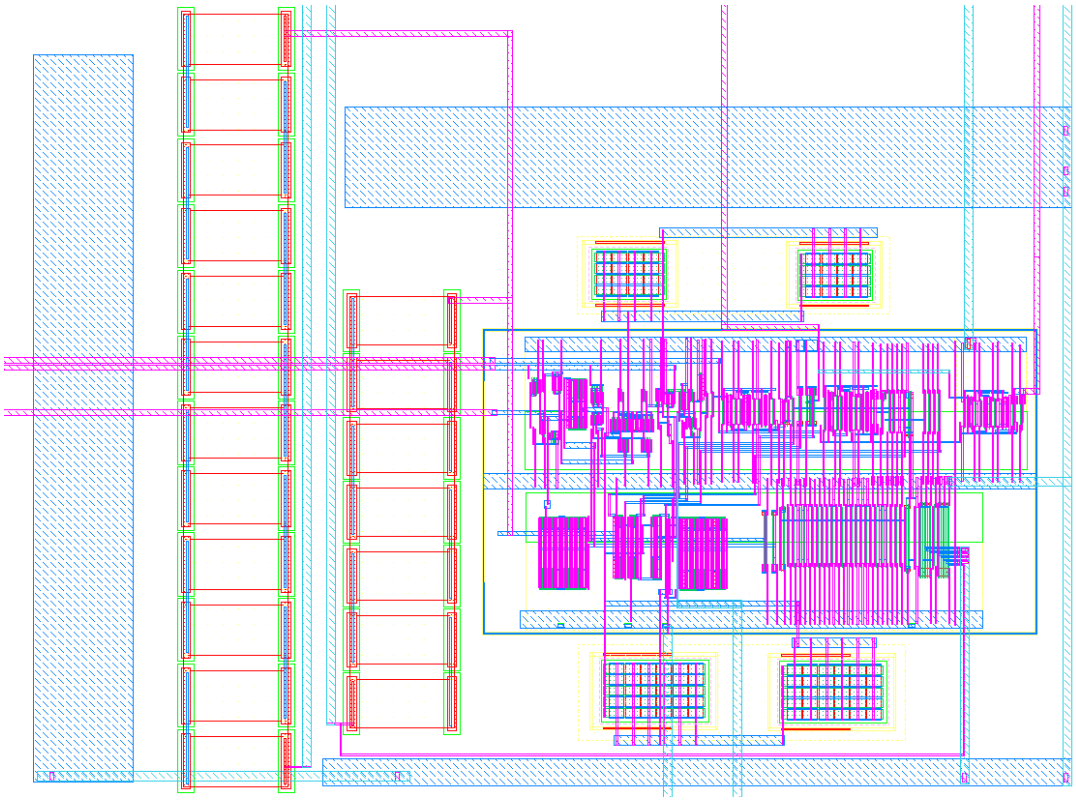

Figure 3.14: Layout of the single resistor with mixed-signal neuron with 65nm technology node.

Thus, the memristors need to be formed with a particular forming voltage depending on the memristive material. After forming, the memristors are programmed to a specific resistance value by again applying a programming pulse for a certain period of time. Hence, from the synapse perspective, this test is used to verify initialization of the memristive synapse. Moreover, this test also analyzes the synaptic connection to the mixed-signal neuron because this involves the memristive synapse connection to the analog neuron. So, this helps in verifying the twin memristive synapse as well as the analog neuron.

This test structure has pins similar to previous tests but also includes pins corresponding to the forming and programming of twin memristors which are V_{formn}, V_{progn}, V_{formp} and V_{progn}. The schematic of this test structure is shown in Fig. 3.15 and the layput is shown in Fig. 3.16. The pin assignment and the details of simulations are included in appendix B.2.

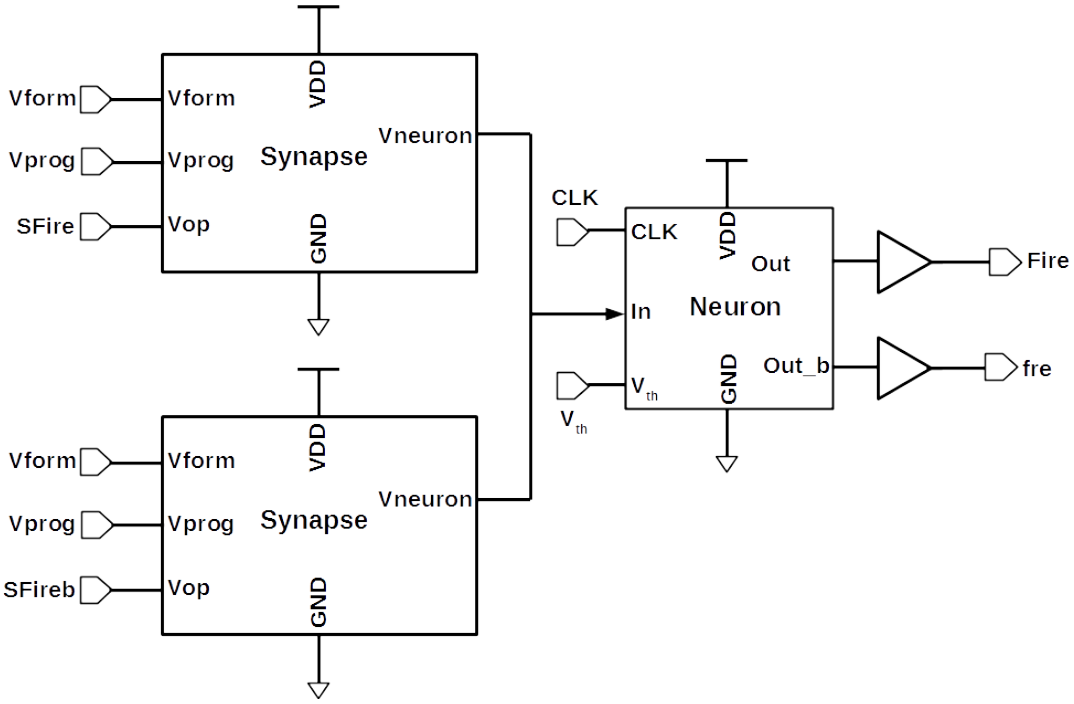

Figure 3.15: Schematic of the single memristive synapse with mixed-signal neuron with 65nm technology node.

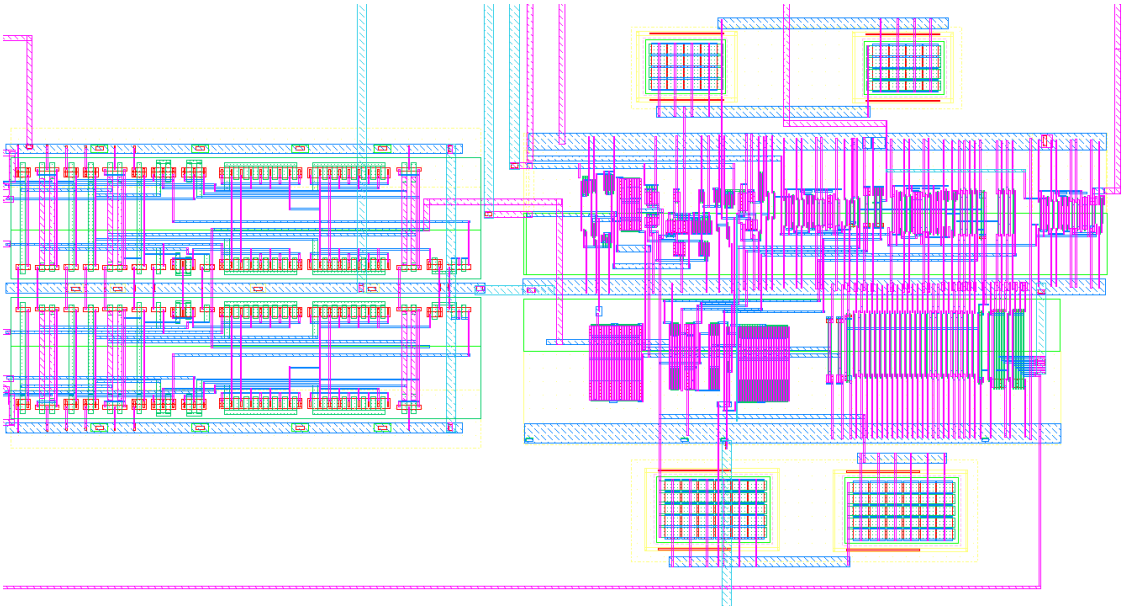

Figure 3.16: Layout of the single memristive synapse with mixed-signal neuron with 65nm technology node.

3.3.4 Single Neuromorphic Core

This test structure helps in verifying the neuromorphic core. A neuromorphic core can be described as a connection of several synapses driving a single neuron. This core acts as the building block of a multiple neuromorphic core processor. A detailed description of the neuromorphic core and the system is provided in Chapter 4. For this test structure, the motivation is to verify the fan in of the neuron and the connection of the synapses. Instead of memristive synapses, this test includes eight resistive synapses shown in section 3.3.2. Each resistive synapse receives pre-synaptic input from incoming pulse signals. The neuron generates an output fire when the accumulated charge is higher than the threshold voltage. So, this test structure has eight input signals connected to the resistances and one output signal as a post-synaptic output.

The schematic of the test structure is shown in Fig. 3.17 and the corresponding layout is shown in Fig. 3.18. The details of this test structure with pin assignment and simulation result with testing strategy is discussed in appendix B.3

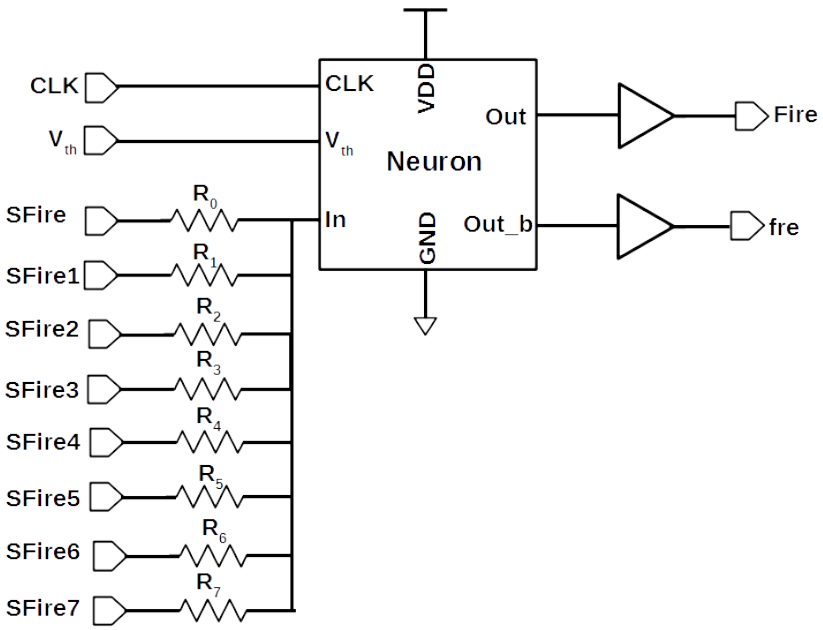

Figure 3.17: Schematic of the core prototype with 65nm technology node.

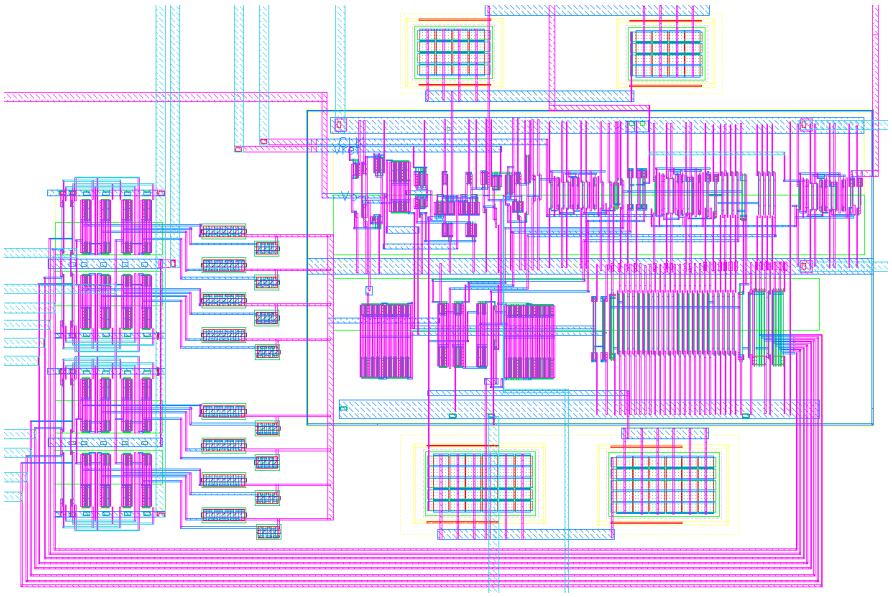

Figure 3.18: Layout of the core prototype with 65nm technology node.

3.4 Synapse and Neuron Energy

The energy efficiency of any system depends on its components. Synapses and neurons are the building blocks for the proposed memristive neuromorphic computing system. Here, the energy for the memristive synapse and the mixed-signal neuron are first determined separately. Then with the component level energy, system level energy is determined. In the literature, some works reported the amount of energy consumption for their neuromorphic components. For example, in [39], the energy consumed by each synapse is $36.7pJ$ for learning where a resistance range of 70Ω to 670Ω for the memristors has been considered. The energy consumed per synapse is $11pJ$ to $0.1pJ$ and in [12] with the working resistance is from 1kΩ to 1MΩ. Considering these existing works, three different types of memristive devices have been considered for the proposed system (see Table 3.2), specifically TaO_x, TiO_2 and HfO_x based on information provided in existing literature.

In this design, we considered four different phases of synapse operation. In Table 3.2, the active phase is when the synapse is actively receiving synaptic input signals. The idle phase is defined based on the inactive phase of the synapse. Potentiation and depression are the two online learning phases when the synaptic weight is increased or decreased respectively based

Table 3.2: Synapse energy with metal-oxide memristors

Synapse Phase \ Devices	TaO_x [114]	HfO_x [104]	TiO_x [65]
	Energy per spike		
Active	$8.074pJ$	$0.48pJ$	$0.17pJ$
Idle	$0.002pJ$	$0.002pJ$	$0.002pJ$
Potentiation	$10.76pJ$	$0.65pJ$	$0.26pJ$
Depression	$10.38pJ$	$0.58pJ$	$0.13pJ$

on the position of the pre- and post-synaptic spikes. If the pre-synaptic spike arrives before the post-synaptic spike, the synapse is in potentiation and if the pre-synaptic spike arrives simultaneous with the post-synaptic spike, the synapse weight is decreased or depressed. For this design, we consider the signle clock cycle before and after the post-synaptic spike considering the DLTP mechanism.

Like the synapse operating phases, the mixed-signal neuron also has three different operating phases and those are idle phase, accumulation phase and finally the firing phase. During the idle phase, the neuron remains inactive meaning it receives no input and experiences minimum activity. During idle, the neuron consumes energy of approximately $7.2pJ/spike$. Next phase is the accumulation phase when the neuron receives the pre-synaptic input spikes and accumulates the charge before reaching a threshold. During the accumulation phase the integrator part of the neuron is active and consumes around $9.81pJ/spike$. And the final phase is the firing phase when the neuron's accumulate charge is higher than the threshold and the neuron generates a post-synaptic spike. This phase includes the cooperator and the digital circuit components to generate a digital pulse and the energy consumed during the firing phase is approximately $12.54pJ/spike$.

These energy data are calculated with a single system clock of $20MHz$. To determine the energy value from the circuit level simulation, the currents through the neuron are sampled for three different phases of operation. Then the average current per spike is calculated from the obtained data and this average current is multiplied with the supply voltage to obtain the average power for each phase. From the average power, the average energy per spike is calculated using the timing duration of each phase.

The energy consumption of the mixed-signal neuron in different phases is summarized in Table 3.3. The calculation of neuron energy considers all the analog and digital components and hence the energy estimation in this work is a bit higher than the pure analog alternatives in the literature. However, the output spikes generated digitally can be routed through the complete system in a more efficient way which will ensure greater drive strength with robust communication. For comparisons against the neuron energy with existing works, there has been energy reported as $6.04nJ$ for $16.67MHz$ in [59] also $8.29nJ$ for $10MHz$ in [58]. There have been other works reported with lower energy such as in [110] where the clock frequency is as slow as $1MHz$. In addition, the operating voltage pulse ranges from "$-100mV$" – "$140mV$" which makes the energy consumption lower but raises concerns for the drive strength of the propagating spikes.

The component level energy estimations are the building blocks for the total energy estimation of the system. The per spike energy estimation of synapses and neurons in Table 3.2 and 3.3 help in estimating the total energy for any application implemented on the system. It is a time consuming and tedious process to determine the total energy of any application for the total system using the low-level circuit design simulator. So, a high-level simulator is used in determining the system level energy details. More details on the high-level simulator for the system level simulation are discussed in Chapter 4.

Here, three different phases for neuron and four different phases for synapses are considered and hence the high-level simulator would track the activity factors for these phases. Activity factors refer to the number of spikes for all the neurons and synapses in these phases throughout the simulation time. Each of these numbers are summed based on what phases they fall into. Then the total activity factor is multiplied by the energy per spike estimation for the corresponding phase. Summing up all the energy values leads to

Table 3.3: Energy consumption of neurons in different phases

Neuron Phase	Energy per spike (pJ)
Idle	7.2
Accumulation	9.81
Firing	12.5

the total energy consumed by the system for the application to complete. This total energy estimation algorithm can be summarized in the following equation 3.15.

$$\begin{aligned} Total\_energy = & \, Energy\_per\_spike_{syn\_idle} \times Number\_of\_spikes_{idle} \\ & + Energy\_per\_spike_{syn\_active} \times Number\_of\_spikes_{active} \\ & + Energy\_per\_spike_{syn\_pot} \times Number\_of\_spikes_{pot} \\ & + Energy\_per\_spike_{syn\_dep} \times Number\_of\_spikes_{dep} \\ & + Energy\_per\_spike_{neu\_idle} \times Number\_of\_spikes_{neu\_idle} \\ & + Energy\_per\_spike_{neu\_accu} \times Number\_of\_spikes_{neu\_accu} \\ & + Energy\_per\_spike_{neu\_fire} \times Number\_of\_spikes_{neu\_fire} \end{aligned} \tag{3.15}$$

This algorithm for energy estimation has been developed in order to build a system where we can estimate the energy at a hardware level. Because energy estimation is a critical factor for designing any system and it is also helpful to have a low-level circuit simulation with energy estimation. Moreover, energy is one of the main motivations where researchers are working hard to build energy efficient systems. That's why, this approach to estimate energy using data from low-level and high-level simulation help in getting an energy estimate for a system.

Chapter 4

Mixed-Signal Neuromorphic System

4.1 Architecture of Neuromorphic System

A neuromorphic system includes synapse and neuron design blocks as the fundamental units. But the placement and connection of these components must contend with several interconnect challenges. Considering these, the proposed neuromorphic system in this dissertation is designed using $m \times n$ memritstive neuromorphic cores, as mentioned briefly in Chapter 3. A neuromorphic core can be defined as a collection of memristive synapses connected to one mixed-signal neuron. This neuromorphic core is specially designed with an aim to achieve the "analog in and digital out" mechanism that makes the computation and connectivity of the proposed system reliable for designing spiking neural networks. The structure of the neuromorphic architecture is shown in Fig. 4.1 which illustrates a system of several neural cores with each core including multiple synapses with one single neuron.

As mentioned earlier, the connection of the synapses and neurons in a neuromorphic system depends on some interconnect issues. For example, the placement of the neurons and synapses are costly in performance because if the neurons and synapses are placed independently instead of simultaneously when laying out a neuromorphic core, the wires connecting the components would be relatively long. The longer the wires are, the larger the capacitance of the interconnects, resulting in lower performance for the system. Moreover, synapses in different locations would experience variation in interconnect capacitance to the neuron and the charge accumulation would also vary even though the synaptic weights

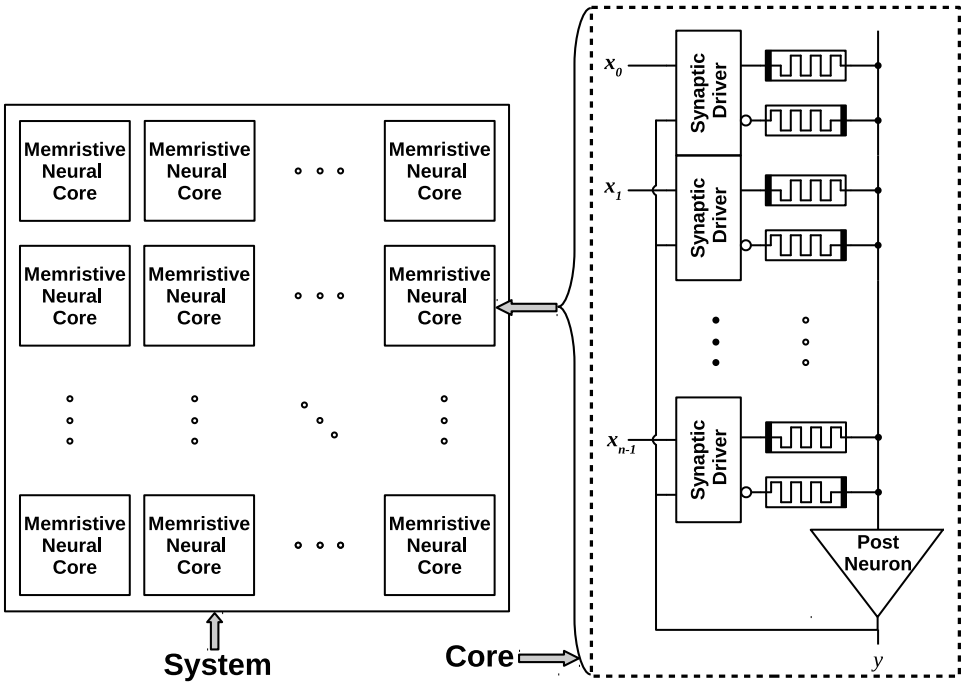

Figure 4.1: A representation of memristive neuromorphic core system [17].

would be same. These issues are of huge importance since the motivation is building a high performance neuromorphic system. Hence, this work presents an innovative configuration considering the performance issues where the synapses and a neuron are placed inside a memristive neural core as shown in Fig. 4.1(right). This configuration ensures a better arrangement of the synapse and neuron so that similar capacitance is maintained across the synaptic outputs to the corresponding neuron. Also, the similar distances between the synapses and the neuron inputs ensure a negligible amount of difference in charge accumulation.

The main goal of this dissertation is to design a neuromorphic system with the memristive synapses and mixed-signal neurons described in section 3.1 and 3.2. This overall architecture is tailored for implementing artificial neural networks. So, it can be described as a specialized hardware for processing neural networks with an emphasis on energy and area efficiency. In addition, the research goal behind this work is to contribute to the community in translating neural networks to circuit-level components so that there is a strong bridge between the simulator and the low-level circuit components. We started drafting this work based on a high-level architecture called NIDA by Schuman *et. al* [85]. NIDA is a continuous time

recurrent neural network architecture which is specifically built as a spiking neuromorphic platform. It includes the biologically inspired feature of dynamic behavior and is also represented in a three dimensional space. NIDA networks are event driven, meaning the networks deal with asynchronous firing or spiking events. The networks are generated using a genetic algorithm called Evolutionary Optimization (EO) and all the networks contain neurons and synapses [87]. NIDA neurons are connected to several synapses on each layer, with each storing charge until a corresponding threshold is reached. Like neurons, synapses are defined within three dimensional spaces as well. All the synapses are determined by the neurons they are connected to. Each synapse contains a synaptic weight which regulates the charge accumulation of the connected neurons. The synapses also include the feature of synaptic delay representation. One of the most interesting features of NIDA is the networks tend to be very sparse and small, yet they have been shown to achieve good accuracy for different tasks such as classification and control problems, often as high as that of conventional deep learning networks [88]. Considering the benefits of the NIDA architecture, we considered a hardware implementation that can accelerate the computational efficiency of NIDA architecture at the hardware level. DANNA in [25] is a hardware implementation on FPGAs based on the NIDA architecture which is robust and almost reaches the efficiency of NIDA. Since we do not have dedicated hardware to explore the promising aspects of NIDA, we began to explore several emerging technologies to build energy and area efficient hardware. This is the primary motivation behind designing the neurons and synapses discussed in section 3.1 and 3.2. Our approach to this research starts with a top to bottom perspective and later moves to a bottom-up approach to verify the system level architecture within the existing software framework. This dual approach provides confidence in building a robust and efficient neuromorphic system.

In order to build a software framework, C++ models have been developed considering the behavior of the memristive synapses. The model captures several memristive features as parameters so that the model is adaptive to circuit level variation. Like synapses, a neuron model has also been developed in C++ which preserves the circuit level characteristics of a mixed-signal CMOS neuron including the current input feature. The system level simulator model also utilizes the online learning mechanism (DLTP) to train and test

networks generated using a genetic algorithm or evolutionary optimization. The following section 4.2 explains more about network training and generation using a genetic algorithm. Also, section 4.3 explains the high-level simulation framework including the synapse and neuron models and system level energy estimation process.

4.2 Network Initialization and Evolutionary Optimization (EO)

Neural networks can be constructed with different topologies where the network size and connectivity vary depending on the topology used. Some topologies work well with classification problems whereas some perform better for control tasks. It is a challenging task to find a topology for a neural network that is suitable for a general set of problems. In this work, a genetic algorithm called Evolutionary Optimization (EO) proposed by Schuman *et. al* in [87] has been utilized for network initialization. EO has been successful in generating optimized spiking neural networks specifically for neuromorphic systems. It works well with basic logic problems as well as classification problems [87] and control tasks [24].

To generate an initial network for any specific application, the genetic algorithm or EO goes through several steps. At first, the user needs to specify the number of input and output neurons. By specifying these numbers, users actually provide EO information about the task (input neurons) and what would be returned back from the network (output neurons). Besides the input and output neurons, the user also specifies an initial number of hidden neurons and synapses. Then a population of initialized networks is generated which contain the same number of input, hidden and output neurons and synapses. The placement of input and output neurons are same for all the randomly initialized network but the hidden neurons and synapses are random, making the networks in the population distinct from each other. Moreover, the connectivity of the network is random as well with the possibility of both feedback and feed-forward connections.

While training a neural network, one important thing that needs to be specified by the user is a fitness function for the specific task. This fitness function can be defined as a metric

to verify the quality of the network since the fitness function receives the network as input and returns a numerical value based on the performance of the network for the particular task. So, the fitness function is used to measure the quality of the networks in the population and helps in scoring the networks so that the best networks would be chosen as parents for the reproduction process in the next step. Usually the better performing networks are selected for producing the next generation by default. When there are parent networks present, crossover and mutation operations are applied in a probabilistic way to generate children networks. Here, crossover means combining sub-networks of parents to produce children networks while mutation refers to making some structural change such as adding or deleting a neuron or changing a parameter such as the threshold of a neuron. After producing the children networks, the fitness evaluation again evaluates the networks and scores those for next step reproduction. In this way, the reproduction, evaluation and selection process is continued until the fitness function reaches a desired value for the particular task of interest. Then the highest performing task is selected and returned to the user to be deployed on the hardware or the simulator with online learning to provide more possibilities for the synaptic weights to be refined. This network initialization and generation algorithm is summarized in Fig. 4.2.

A genetic algorithm is very helpful in producing optimized networks for a variety of tasks given certain constraints. For instance, it has the ability to perform well with synaptic weight constraints of the memristive devices and also constraints on the network connectivity. Unlike other fixed topologies, genetic algorithms optimize the network at its best possibility within the constraints of the system instead of mapping the ideal parameters to the reality. Also, it can operate with a software simulator as well as "chip in the loop" for evaluation. Another interesting feature of the model used here is that the programmable synaptic delay can be easily programmed with the genetic algorithm. This is done by mapping the network in a 2-dimensional grid where the distance between the synapse and corresponding neuron represents the synaptic delay. Moreover, these delays can be altered using mutation to produce more optimized and efficient networks.

```
1: procedure EVOLVE
2:     population = INITIALIZEPOPULATION
3:     MaxFitness = -1
4:     epoch = 0
5:     while MaxFitness < and epoch < MaxEpoch do
6:         fitnesses = []
7:         for net in population do
8:             fitnesses[net] = FITNESS(net)
9:             if fitnesses[net] > MaxFitness then
10:                MaxFitness = fitnesses[net]
11:                BestNet = net
12:            end if
13:        end for
14:        children = []
15:        while size(children) < size(population) do
16:            p1, p2 = SELECTPARENTS(population, fitnesses)
17:            if RANDOMFLOAT < CrossoverRate then
18:                c1,c2 = CROSSOVER(p1,p2)
19:            else
20:                c1 = DUPLICATE(p1)
21:                c2 = DUPLICATE(p2)
22:            end if
23:            if RANDOMFLOAT < MutationRate then
24:                MUTATE(c1)
25:            end if
26:            if RANDOMFLOAT < MutationRate then
27:                MUTATE(c2)
28:            end if
29:            children.append(c1)
30:            children.append(c2)
31:        end while
32:        population = children
33:        epoch += 1
34:    end while
       return MaxFitness, BestNet
35: end procedure
```

Figure 4.2: Network initialization with genetic algorithm [17].

4.3 Software Framework on Low-level Design

An important motivation for leveraging neuromorphic computing is energy efficient hardware specialized for complex neural computations with feature like parallel processing. For this, the design needs to be verified from various perspectives. For instance, both the hardware and high-level networks need to be compatible so that the simulator is aware of hardware details. Hence, there is need for a software simulator which will model the neuromorphic hardware as accurately as possible, bridging the gap between the simulated network and the hardware itself.

A software stack has been developed by the TENNLab research group at UTK to work with a large range of neuromorphic systems. This software repository is helpful in connecting different neuromorphic algorithms such as NIDA, DANNA and mrDANNA (our

memristive neuromorphic system). Among these three, mrDANNA networks are based on the neuron and memristive synapse models described in this work. To build a software stack for this particular memristive neuromorphic system, the models have undergone multiple design iterations since the neuron and synapse models have been evolving from the very beginning. Currently, the models used in the simulator contain the latest versions of the equations and parameters that best relate to the represented hardware. To make the software stack consistent with the design, there are connections among the architecture, learning, application and the software stack (Fig. 4.3). The software stack works by training a neural network using a genetic algorithm and can generate networks for specific applications, particularly for memristive neuromorphic architecture. The stack is also responsible for simulating the generated network and can be used to estimate the energy consumption for the application.

4.3.1 High-level Synapse Model

Designing the synapse model for the memristive neuromorphic software simulator, several details have been incorporated from the hardware specification and into the behavioral model. For instance, hardware synapses include twin memristors with parameters of high resistance state (HRS), low resistance state (LRS), switching time in positive and negative

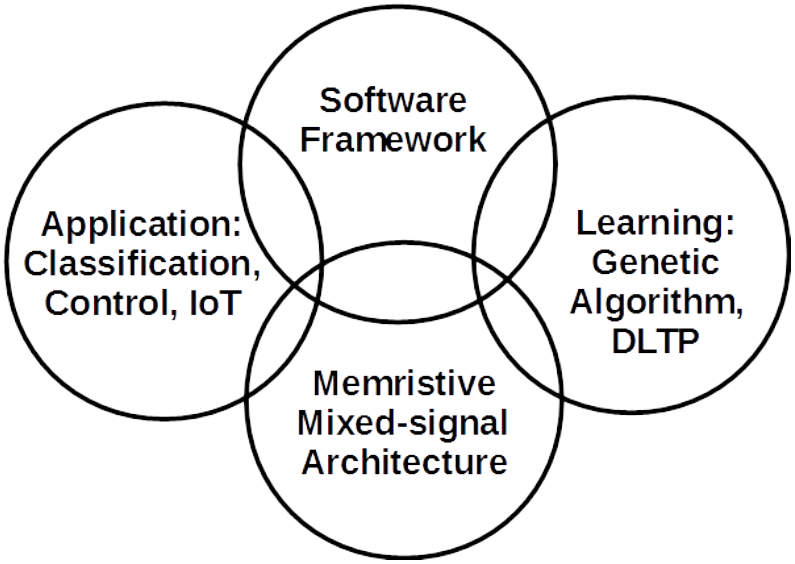

Figure 4.3: Relation of software framework with architecture learning and application.

directions, and switching voltages. These memristive features are added to the simulation model in order to provide an accurate representation of memristor based weight updates. Additionally, the synapse model also includes synaptic delays (from 1 to 7 cycles) which are present in the hardware synapse component as a delay chain. This model also has a parameter for initializing the number of unique resistance states possible for the memristors during training. The interesting feature of the twin memristive synapse described in section 3.1 is that the synapse receives input as voltage pulses and then supplies outputs as weighted currents. The software model is also tailored in such a way that the synapse node will take voltage input as an event and generate output current for the neurons. This way, the software simulator is essentially a circuit level simulation while training and testing, but only for analog components such as the twin memristor structures. The model defines the critical parameters of the synapses, particularly the analog sections have been detailed in the high-level model by using similar equations and behavior from the low-level design. However, other sections of the synapse circuits, such as the digital logic blocks, are kept as abstract. This way, the simulator model captures the important details and also accelerates the simulation as compared to low-level circuit simulators.

4.3.2 High-level Neuron Model

The neuron design considered for the hardware implementation of this neuromorphic system is based on integrate and fire mechanism described in section 3.2. According to the neuron characteristics, the mixed signal neuron accumulates incoming charge until a certain threshold is reached. To be more specific, the neuron receives weighted current inputs from the synapses connected to its input and then integrates the corresponding charge. When the accumulated charge is higher than the threshold, the neuron generates a firing event in the form of an output pulse or spike. While designing the neuron model for the high-level simulator, hardware features such as current inputs, threshold voltage, integrator feedback capacitance and also the voltage output were considered as parameters. All of these are arranged in the neuron model so that its performance matches with the hardware component. Another interesting feature of the neuron model is that it has a parameter called "STDP cycle length" which can be changed by the user depending on what type of online learning

mechanism is desired for the network simulation. Here, the analog components have been modeled in detail, including the integrator capacitance and the input analog current from the synapses. Like the synapse model, the other digital circuit components of the neuron are kept as abstract to make the high-level model faster. However, obtaining precise data for sensitive parts, the model ensures accuracy in cycle to cycle verification. Online learning in EO based initialization is discussed in a later section.

4.3.3 Verification of High-level Simulator Testing

The high-level simulator is built not only to train memristive neural networks but also to simulate and test the network. To rely on the simulator, it is verified against simulation results from low-level circuit simulator, specifically Cadence Spectre. As a verification aid, the high-level simulator produces a detailed event log that includes the result of each neuron and synapse firing event in cycle to cycle precision. Hence, this simulator is defined as a "*cycle accurate*, *event driven*" simulator. The outputs and event logs from the simulator are also used to produce images of any given network simulation.

For verification of the high-level simulator compared to low-level circuit simulator (Cadence Spectre), a small classification network has been chosen and simulated using both simulators. The network selected for this task is from iris flower classification dataset [55]. The dataset contains 150 test-cases for three classes of iris flowers where each case includes four features of a flower. Chapter 5 provides more detail about this dataset. During the verification process certain assumptions were made for computing purposes. For example, the input and output neurons are assumed to be connected through non-learning synapses. Also, a single test case is leveraged so that the run time is reduced. The inputs are processed and programmed following the rules provided by neuromorohic library with those used by both simulators to start the verification process.

The network used for this test has been generated using genetic algorithm with the resulting network having seven input neurons, three output neurons and a single hidden neuron. Since, the hidden neuron has no output connection, this is apparently inactive and can be pruned. This inactive neuron is included to illustrate the random nature of the genetic algorithm approach to training. The network is shown in Fig. 4.4. The synapses are

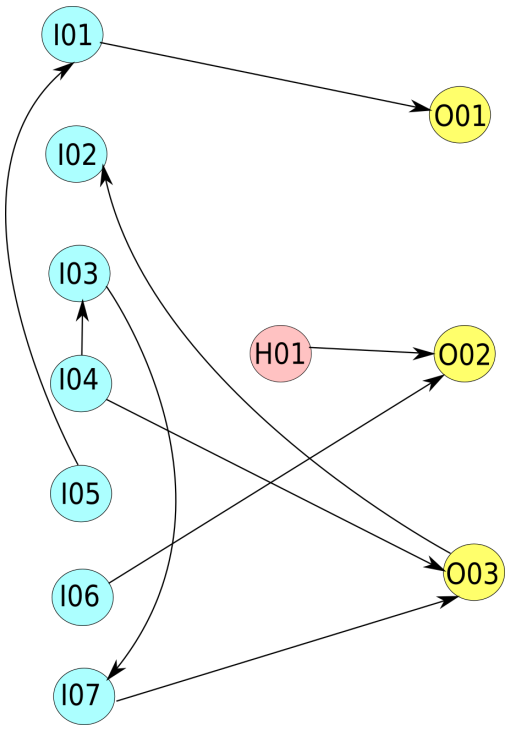

Figure 4.4: An example of Iris network [94].

denoted by the arcs with direction indicating information flow and connectivity. Here, the input neurons are marked with an 'I' and the output neurons are marked with an 'O'.

One of the important metrics in performing the verification test is the run-time of each simulation. It is determined using high precision timers built into the operating system with the same configurations, specifically a 4th generation Intel i7 processor in this case. The low-level circuit simulator Cadence Spectre produces an output graph after the simulation. A Python script is used to process the event log from the high-level simulator to generate corresponding output graphs. Both outputs as well as events are verified against one another to ensure that all the events such as firing, delay, and accumulation occur at the same cycle for both simulators. This way, the high-level simulator justifies its cycle accurate event driven nomenclature.

Fig. 4.5 and Fig. 4.6 show results from the high-level and Cadence simulations respectively. Here, the input spikes are shown for input neurons two, three and seven since the input pulses are received on those three neurons only. Among three output neurons, only the third output neuron fires twice determining the iris flower class as Virginica.

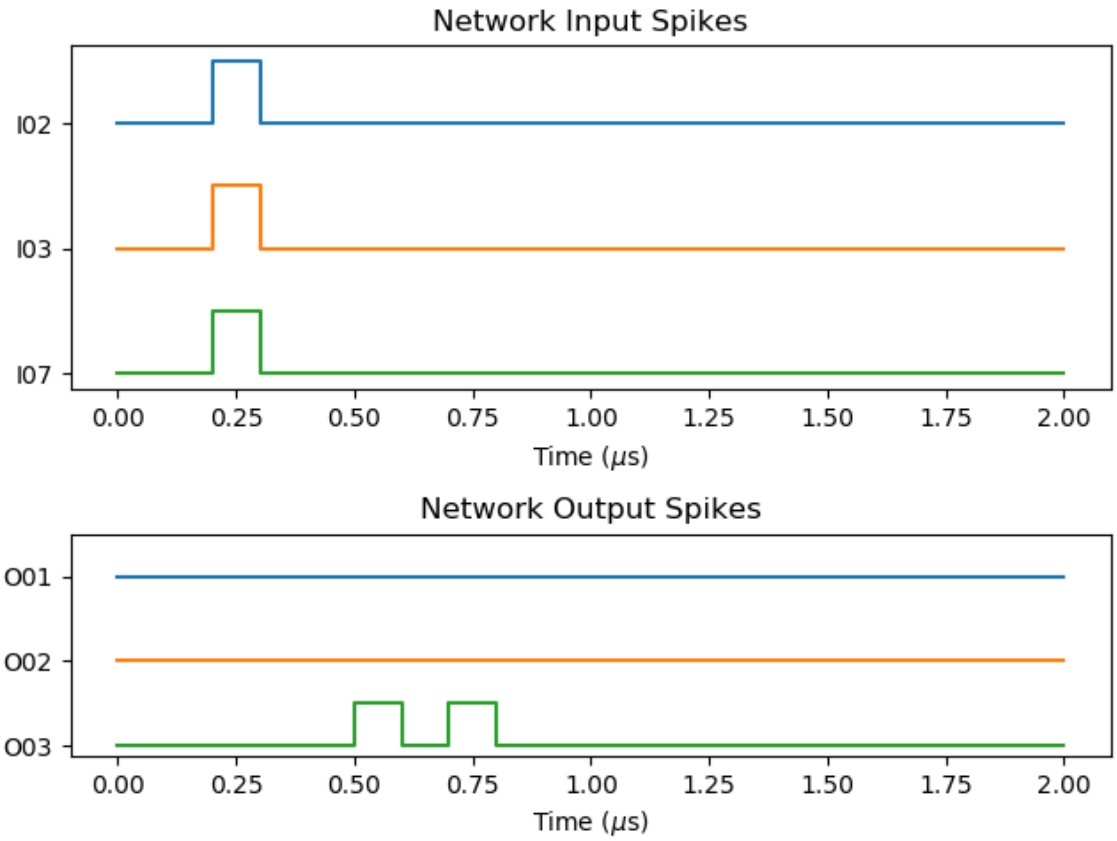

Figure 4.5: Inputs and outputs for Iris network in high-level simulation [94].

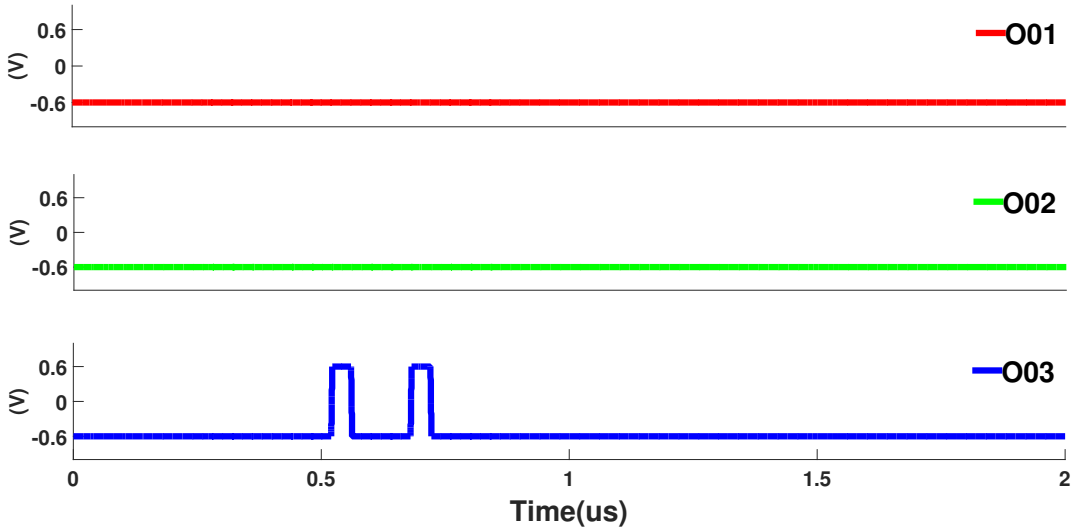

Figure 4.6: Inputs and outputs for Iris network in cadence simulation[94].

Taking a precise look at the figures, it can be said that the low-level Cadence simulator is extremely time accurate in detailing events whereas the high-level simulator is cycle accurate in detaining events. This helps in processing events at each cycle and then grouping them so that similar groups can be simulated in batch. This way, the simulation speed can be further improved. In fact, the runtime difference for high-level and Cadence simulations is very large, 632.6 seconds in Cadence using 8 processing cores and 5 milliseconds for the high-level simulator on a single core. This illustrates the efficiency of the high-level simulator which is able to log the event details with fast and accurate results.

4.3.4 High-level Energy Estimation

Since the synapse and neuron models in the high-level simulator have identical features as those in the hardware circuit components, it is easier for the high-level simulator to provide interesting insights about the hardware without simulating a network at the circuit-level, using Spectre or SPICE. This provides an advantage when estimating total energy consumption and some process variations before the hardware is fabricated. Hence, it helps in analyzing the neuromorphic system for further improvement. Moreover, from the software perspective, using realistic the synapse and neuron models help with training the neural networks in an energy efficient way.

The energy estimation of the overall neuromorphic system has been described in section 3.4. In that section, the energy per spike for each synapse and neuron is determined and with those values used to help in determining the energy consumption of the whole system. This process for energy estimation needs extensive manual tracking of activity factors in the network along with some manual calculations using equation 3.15 to calculate the total energy consumed by the system. Considering the manual work, the high-level simulator is designed to do the extensive calculations and is now able to provide the energy estimation after each network simulation.

The algorithm for high-level energy estimation from the simulator is shown in Fig. 4.7. The way the simulator determines the energy is very similar to the manual computation. The only difference is that the user does not need to manually keep track of all network events. The simulator recognizes every single event, such as pre-synaptic input fires, post-synaptic

```
1: procedure ENERGY_ESTIMATION
2:     Num_neu = Number of neurons
3:     Num_syn = Number of synapses
4:     Num_cycle = Number of cycles
5:     neuron_fire = []
6:     neuron_accumulate = []
7:     neuron_inactive = []
8:     synapse_fire = []
9:     synapse_potentiate = []
10:    synapse_depression = []
11:    synapse_delay = []
12:    synapse_inactive = []
13:    while event do
14:        if neuron_phase == firing then
15:            neuron_fire ← neuron_fire + 1
16:        end if
17:        if neuron_phase == accumulation then
18:            neuron_accumulate ← neuron_accumulate + 1
19:        end if
20:        if synapse_phase == firing then
21:            synapse_fire ← synapse_fire + 1
22:        end if
23:        if synapse_phase == potentiation then
24:            synapse_potentiate ← synapse_potentiate + 1
25:        end if
26:        if synapse_phase == depression then
27:            synapse_depression ← synapse_depression + 1
28:        end if
29:        if synapse_phase == delay then
30:            synapse_delay ← synapse_delay + 1
31:        end if
32:    end while
33:    neuron_inactive ← Num_neu × Num_cycle − (neuron_fire + neuron_accumulate)
34:    synapse_inactive = Num_syn × Num_cycle − (synapse_fire + synapse_potentiate +
       synapse_depression + synapse_delay)
35:    energy_neuron ← Σ energy_per_spike × neuron_phases
36:    energy_synapse ← Σ energy_per_spike × synapse_phases
37:    energy_total ← energy_neuron + energy_synapse
       return energy_total
38: end procedure
```

Figure 4.7: High -level energy estimation algorithm.

output fires, weight update in both potentiation and depression, accumulation, and firing, that occurs on the synapse and neuron models. Each of these events are accounted for as activity by the simulator which counts the activities from the beginning of any simulation. If there is no activity on the models, it also keeps track of that as inactive or idle phases. At the final stage, the net number of activities is determined by subtracting the total activities from the inactive events. It must be mentioned here that the energy per spike values for each operational phase of the synapses and neurons are parameters for the high-level simulator. Here, the energy per spike values of each phase is provided to the high-level simulator. Hence, it does not need to calculate all the energy values from the current and voltage equations.

Thus, it helps speed of the simulator to be faster by a factor of roughly 10^5 than the hardware simulator. Basically, it keeps track of energy on each phase and hence the total energy of the system for any network can be easily calculated using the software simulator.

To verify that the energy estimation from the high-level simulator is similar to the energy estimation from Cadence Spectre, the same iris network has been used. For this small network both the simulators reported total energy consumption of $7.45pJ$ for one single classification.

4.4 Online Learning on High-Level Initialization

Online learning is an important feature in the memristive neuromorphic system considered here. This method helps the network learn using live updates of synaptic weights that influence future decisions based on current experiences instead of relying entirely on a fixed training environment. In this work, we use DLTP (discussed in section 3.1.3) as the online learning method according to which the synaptic weights are updated based on the relative position of pre- and post-synaptic spike events. DLTP is incorporated into the high-level simulator during the testing process so the system learn from unknown environments. However, DLTP has been utilized for offline training as well. To be specific, DLTP helps in altering the synaptic weights while measuring the fitness of the network. This actually provides the opportunity to assess network fitness and choose a comparatively better network for further training. For instance, the iris classification task considered here requires 22500 cycles for fitness evaluation and also involves many classification events that represent a whole training epoch.

The DLTP mechanism is a part of the synapse model in the high-level simulator. A brief description of high-level synapse model is presented in section 4.3.1 where it is mentioned that the synapse model has a parameter for initializing the number of unique synaptic weight states. To elaborate on this feature, it can be defined as a mapping of the resistance of the memristors to some abstract weight values. By doing so, the simulator gets the opportunity to explore a specified range of abstract synaptic weight values while training any network. Here, the twin memristive model is considered to have a symmetric range assuming the

highest positive weight would have the same magnitude as the highest magnitude negative weight. This is advantageous for generating a network for exposure to DLTP. However, the synaptic weight here is initialized to some integer value, even though those can be anywhere between the range while being trained on DLTP. Hence, the networks can be restricted by the genetic algorithm while allowing online learning to modify and fine tune the synaptic weights for improved results.

The resistance mapping to an abstract weight follows some cumulative steps. The first step is to represent the largest effective conductance of the synapse as the largest abstract weight. After that, the effective conductance representing the abstract weight of 'one' is determined. Then the effective conductance for the weight of 'one' can be utilized in normalizing any effective synaptic weights present in the neural network. Since DLTP is used in the synapse model, synaptic weight updates due to DLTP will affect the resistance value updates of both memristors in a twin memristive synapse. So, the memristor values are updated accordingly if there is any potentiation or depression event and later the model updates the effective synaptic weight by updating and normalizing the effective conductance. It can be noted that synaptic weights are related to the effective conductance of the synapses in the model.

Since the effect of DLTP is important for training an optimized network, it should be enabled while training. Moreover, the effects of DLTP on the network depend on the topology of the network since the potentiation and depression of any synaptic weight is determined by the network's connectivity. This is why DLTP is "turned on" during network training using evolutionary optimization. While enabling the DLTP mechanism, there will be networks that have positive effect over DLTP and those networks will show better performance in terms of fitness. On the other hand, there will be networks where DLTP would have a negative impact and will lower the fitness of the networks. If DLTP is disabled during training, it might be possible to generate networks without knowing the adverse effects of online learning on the networks and their performance would degrade while testing. Hence, DLTP is suggested to be enabled during training with a genetic algorithm, considering the long term effects of the network's performance. Some results for DLTP during training and testing are discussed in Chapter 5.

Chapter 5

Application and Results

Our one of the main goals of this research is to determine the performance of the proposed neuromorphic system in terms of potential area and energy efficiencies. For that, we have explored several applications using the software framework and have observed some promising results. We begin with very simple gate-level computations such as XOR and AND operations before moving to larger applications for classification tasks, control applications and high energy particle detection. For each application considered, we determine estimations of the accuracy and energy consumed while the system is running. In Chapter 4, top-down and bottom-up approaches are described for designing the system, keeping the hardware design in close alignment with the software framework. Here, we concentrate on the bottom-up approach which provides a foundation for simulating larger networks using high-level simulator. We are inclined to use the high-level models of our circuit level models as the larger networks are slow to simulate using the low-level simulator. Thus, we have developed the high-level models based on circuit-level parameters to obtain outputs faster but with comparable accuracy.

5.1 Classification Application

Like other computing algorithms, classification applications have been implemented using the proposed neuromorphic system. We focused on total energy consumed for each classification

since energy is one of the main metrics for quantifying system efficiency. Also, we calculate the accumulated accuracy for different applications based on the proposed DLTP learning.

For classification applications three different classification tasks are considered from UCI Machine Learning Repository [55]. Those are iris flower dataset, Wisconsin Breast Cancer dataset and the Prima Diabetes dataset. All of these are commonly used in the literature as benchmark applications for machine learning systems. The iris dataset is a set of 150 flower instances with each instance consisting of four properties of iris flowers. The breast cancer dataset includes 699 instances with each instance defining ten different features of a cell nucleus. Finally, the diabetes dataset includes 768 instances with each defining four different fields per record. All of these datasets have been processed to make them acceptable as inputs to a neuromorphic system. Specifically, the input values have been encoded as integers between 0 to 10 by scaling the raw data such that it is easier to perform a computation using our approach. For instance, an example network for the iris dataset is shown in Fig. 5.1. This network is generated from EO using the genetic algorithm mentioned in Chapter 4. This network includes $four$ input neurons for four features, six hidden neurons and *one* output neuron. The single output neuron represents the output class. The other two applications lead to similar networks generated from EO with input and output neurons defining the input features and classes, respectively. Table 5.1 summarizes the three datasets used in this work.

Since energy is one of the prime metrics for the efficiency, the total energy consumed while each classification is calculated using the calculation algorithm described in Chapter 3. Here, the activity factors for all the neurons and synapses for an application are monitored and stored during the task simulation for different neuron phases (idle, accumulation and

Table 5.1: Characteristics of dataset [55]

Data Set	No. of instances	No. of inputs	No. of Output Class
Iris	150	4	3
Wisconsin Breast Cancer	699	10	2
Prima Indian Diabetes	768	8	2

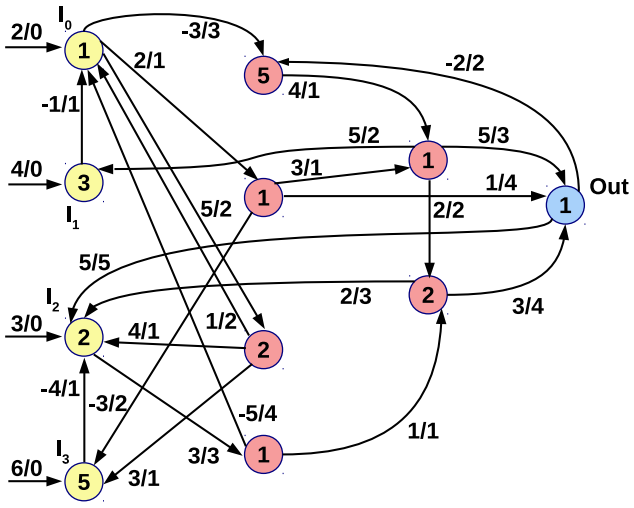

Figure 5.1: An example network for the iris classification task. The input neurons are yellow, hidden neurons are red and the output neurons are blue. The neurons are labelled with their thresholds and the synapse labels denote the synaptic weights followed by the delays [17].

firing) and synapse phases (active, idle, potentiation and depression). The energy per spike for the neurons and synapses are then multiplied with the total activity numbers for all phases. Summing all of these numbers yields the total energy for the classification task. To analyze different suitable memristive devices for the system, the energy estimation is shown in Fig. 5.2 based on three different memristive devices, defined by their LRS and HRS values (LRS/HRS).

Another metric considered here is the effectiveness of using the online learning mechanism. DLTP mechanism described in Chapter 3 is used here for online learning. Networks have been trained both with and without online learning using the genetic algorithm. Those networks are then tested for two cases, either keeping DLTP *on* or turning it off. The accuracy for each classification application has been determined and is shown in Fig. 5.3. To be more specific, the first two columns of the figure show the accuracy of the networks both trained using DLTP but the average accuracy is higher for the network when online learning (DLTP) is present.

Another interesting case has also been considered when the networks are trained without online learning but DLTP is present while testing. Results for this case using all three datasets are also shown in Fig. 5.3 on the third column. This shows that the change

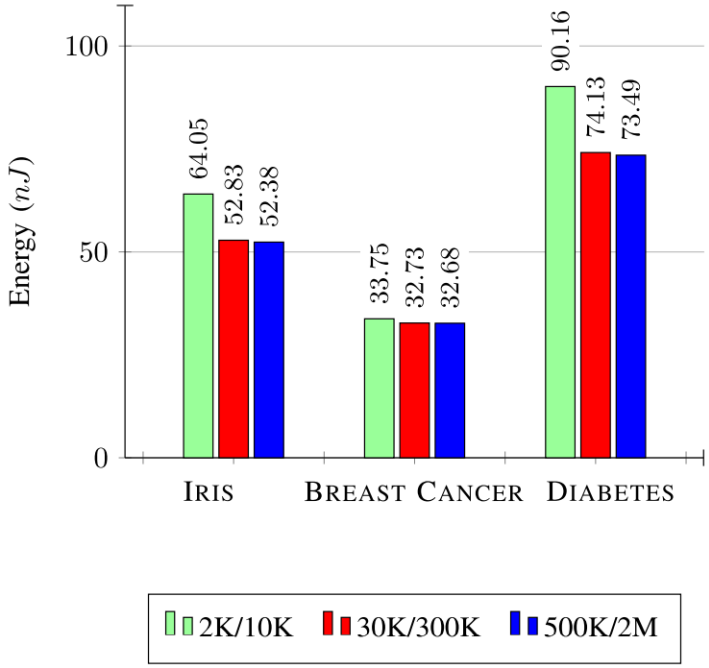

Figure 5.2: Total energy per classification [17].

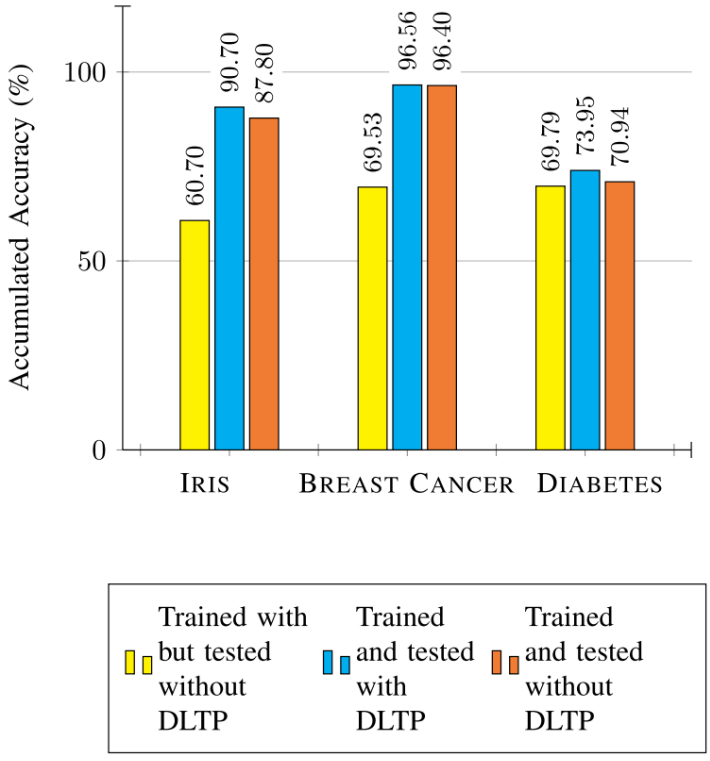

Figure 5.3: Average accumulated accuracy for classification task for network trained with learning but tested with/without learning and trained/tested without learning [17].

in the accuracy result between the networks trained/tested with/without DLTP is very small. However, this result can be justified by considering the average number of epochs to achieve the observed accuracy. Table 5.2 shows that the average number of epochs while training and testing with DLTP is higher than that without DLTP since EO is engaged in numerous iterations for DLTP to reach the highest accuracy with optimized steps. Hence, the DLTP process can be helpful in classification tasks to achieve higher accuracy while training networks. However, DLTP during training is essentially an additional fitness objective which require more epoch to train for as compared to the case with no online learning. In addition, the average accumulated accuracy for all the classification tasks mentioned in Fig. 5.3 is higher for trained/tested with DLTP which is very similar to [93] where an RRAM model is used for simulation and an accuracy of 85% has been reported for iris classification with online learning.

For this work, DLTP has been used as the online learning mechanism. To compare the area efficiency of DLTP, other techniques have been considered from the literature. A very similar technique is a digital implementation of STDP [14] that has been analyzed for two OR gates, two AND gates and a shift register. On the other hand, for DLTP, a driver logic block and an output control block are used which include three NAND gates, two inverters and a flip-flop, as shown in Fig. 3.3. Hence, the DLTP approach is more efficient in terms of area usage. Moreover, the implementation in [14] is accomplished using a Xilinx Spartan FPGA leveraging several LUTS to build the STDP logic.

Table 5.2: Average number of epochs to achieve accumulated accuracy [17]

Data Set	Trained and tested without DLTP	Trained and tested with DLTP
Iris	194.2	267.2
Wisconsin Breast Cancer	37.7	108.6
Prima Indian Diabetes	299	299

Another interesting approach described in [9] also utilizes an FPGA but with block RAMs, a multiplier and LUTs for a successful implementation. Both of these mentioned logic implementations of STDP require LUTs. Hence, the DLTP approach using 65nm CMOS 65nm is more efficient in both energy and area consumption.

One more recognized dataset has been utilized in this work to explore system-level efficiency. MNIST image classification is one of the more popular datasets for handwritten digit recognition. EO has been used to generate networks for MNIST image classification on the proposed neuromorphic system. The network considered here has an accuracy of approximately 90% which is comparable to other non-convolutional spiking neural network approaches such as [26]. The network considered is specifically used for classifying the zero digit. Like other classification tasks, the energy of this classification is also calculated with an operating clock frequency of $16.67MHz$. The average power and energy consumption for one classification task here is $304.3mW$ and $18.26nJ$, respectively. It can be noted here that these power and energy values include both analog and digital circuit components such as delay components and registers. However, the core analog power and energy estimation is much lower at approximately $87.43mW$ and $5.24nJ$ per spike, respectively. These values are comparatively more efficient than other MNIST classification approaches using GPU, FPGA or even ASIC architectures which have power estimations reported in 'W' range [31], higher than other neuromorphic implementations such as IBM's TrueNorth [115].

5.2 Control Application

The internet of things (IoT) is becoming one of the top technologies where almost all the devices are resource constrained and in need of emerging technologies that ensure energy and area efficiency. Hence, memristive neuromorphic computing can be an excellent resource for developing the IoT sector and memristor based spiky neural networks can be be leveraged for IoT based machine learning options. For instance, an autonomous robot and its navigation system is frequently used in IoT control applications. Control applications for robots are usually very resource limited because higher energy batteries also lead to increased size and weight. So, it is preferred to use area and energy efficient batteries. In this work,

a navigation robot described in [69] has been evaluated for the memristive neuromorphic system considered. According to the authors of [69], the robot gets input spikes from the sensors and output spikes are used to directly control the motors. The input sensors used in this task are LIDAR sensors on a servo that takes five measurements in an arc and limit switches which generate input spikes for the robot network. The robot is designed to explore as much space as possible with introduced difficulties and the possibility to adapt to unknown environments using online learning.

This control application network has been generated using the same EO framework with possible room configurations. Each robot navigation simulation is evaluated to make sure the robot performs well in unknown environments avoiding obstacles. For training purposes, the neuromorphic system has been simulated many times instead of the actual physical robot in real environments. An example simulated path is shown in Fig. 5.4. The simulated network is then deployed into DANNA, another FPGA based neuromophic architecture where it has been used to control a physical robot as described in [69].

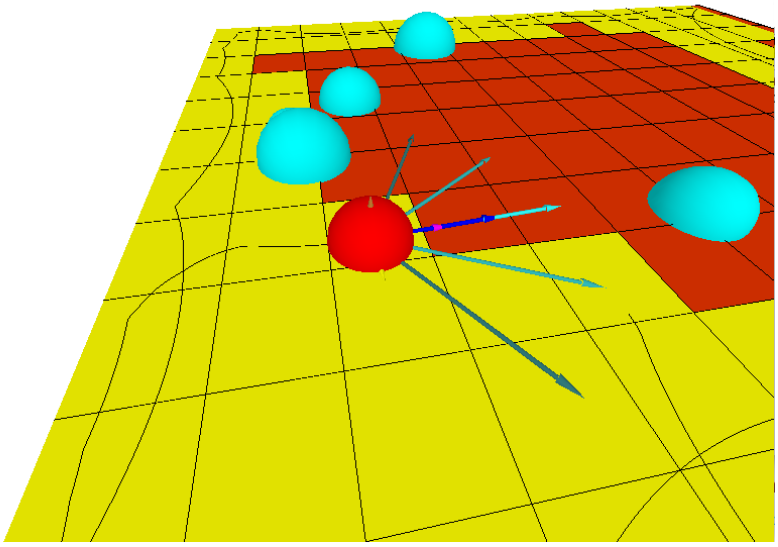

Figure 5.4: Visualization of the robot navigation application. Here, the floor is represented as a grid where the red boxes denote the unexplored section and the explored area is in yellow. The robot is represented using a red sphere and the five blue rays represent its sensors. The obstacles are represented with teal. Robot's taken path is referred with the black path on the floor [19].

The network used in this control application includes 9 input neurons where five of the inputs are from the LIDAR sensors, two other inputs are supplied from the limit switches and the rest are from bias and random values to help with drive functionality. There are 18 hidden neurons and 4 output neurons to control the motion of the motors. The example network is shown in Fig. 5.5. In total, the network includes 119 synapses for communication from neuron to neuron. It shows that the network has a single layer only which makes this representation easy for processing and hence low energy. Specifically, this type of network representation ensures a much smaller network with lower energy consumption as compared to traditional deep learning networks.

In order to analyze the performance of this network, different activity factors for all the neurons and synapses have been recorded. Using the measurements in Tables 3.3 and 3.2, an average power estimation of the network on the physical chip has been defined. An interesting analysis of the total number of spikes present in the simulation shows that the network has an average of 4425 spikes per second but in real time, the robot remains idle most of the time with the vast majority of the spikes becoming trivial for the energy calculation. The network has been simulated for a 20MHz clock where the robot is active while taking decisions only five times per second. So, the average power used by the network is approximately 142.7μW as shown in Table 5.3. The average power reported here is measured only for the core logic of

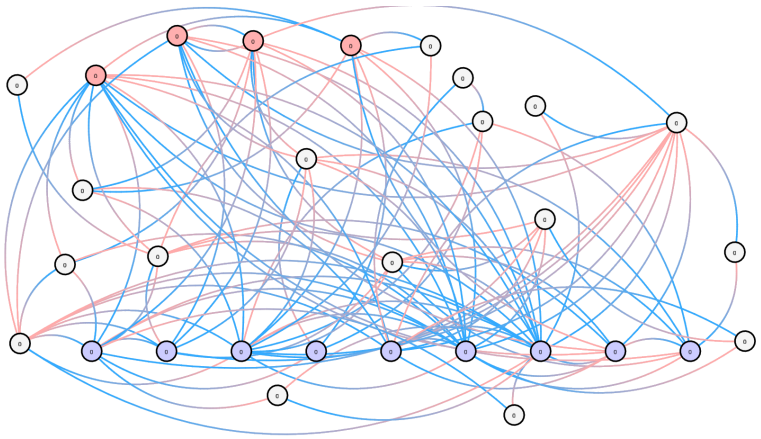

Figure 5.5: An example of robot navigation network. The colored circles represent neurons: Blue refers to input neurons, red refers to output neurons, and white denotes hidden neurons. Synapses are presented by arcs with blue end being the pre-neuron and the pink end being the post-neuron [19].

Table 5.3: A description of a NeoN network

Number of Neurons	31
Number of Synapses	119
Average Spikes per Second	4425
Power Usage (Core Logic)	142.7 μW

the neuromorphic system since most of the energy consumed for computation occurs in the core logic. Again the average power can also be translated into average energy consumption using the clock frequency and here the energy consumption is $7.135pJ$.

5.3 High Energy Particle Application

Somewhat different from other classification and control tasks, we have also worked to apply the proposed architecture to a completely different application: A neutrino particle detection problem using data from Fermi National Accelerator Lab. The task involves the classification of a horizontal region where the interaction between a potential neutrino particle and a projector occurs.

One network example for neutrino data includes 50 input neurons and 11 output neurons. Each output neuron corresponds to 11 class labels in the neutrino data. For the experimental setup, only a single view of the data (x-view) has been considered (shown in Fig. 5.6). The data input in this experiment is different than other applications. The data has been fed as time lattice data instead of using it as an image because the time lattice data carries the time at which the energy values exceeded the threshold. These times are incorporated with spikes to generate the neural network. This results in a network with 90 neurons and 86 synapses which is smaller than networks built using the conventional algorithms, specifically deep learning. This network has been tested with an accuracy of 80.63% which is comparable to the network of 80.42% accuracy trained for a deep neural network where the data there was also restricted to a single view [99]. The total energy for this application has been analyzed and determined to be approximately as $1.66\mu J$ per classification.

This application basically shows the strength of spiky neural networks for classifying spatio-temporal data over deep neural networks. Leveraging the advantage of small and

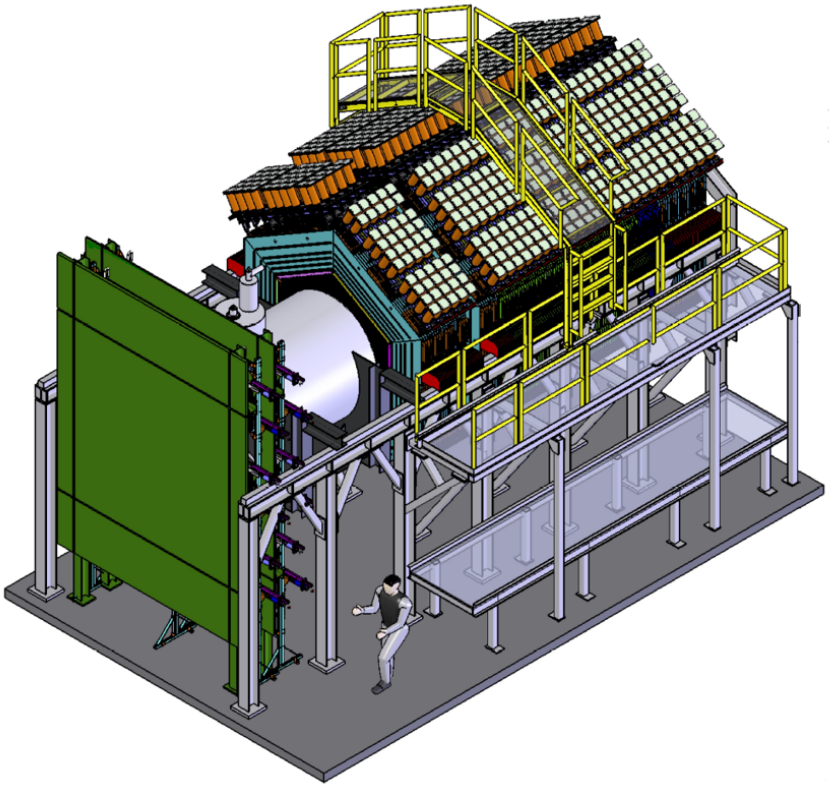

Figure 5.6: MINERvA detector [99].

sparse network generation for this proposed system, it helps in achieving low energy in architectural level. Thus, the proposed neuromorphic system can achieve similar or better accuracy relative to deep learning but with much less area and energy.

Chapter 6

Mixed Signal Neurons with Stochasticity

Considering neuromorphic computers are biologically inspired, this work also considers how the probabilistic characteristics of the human brain can be emulated in artificial systems. Recently, there have been some works on probabilistic approaches such as Bayesian computing with networks that can be used for biologically-plausible implementations of Boltzmann machines and deep belief networks. The neuromorphic system considered here are constructed from IAF or LIF neurons that are more deterministic with few explicit stochastic effects.

Stochasticity can be introduced into IAF neurons via a variety of mechanisms. One simple method is to inject noise into the neuron using incoming signals [78]. However, this process can cause huge increase in power consumption and there is limited availability for scalable features. Some other ways of injecting noise in neurons include the use of noisy firing thresholds and noisy reset voltages. The idea here is to modify the existing IAF neuron so that, irrespective of the incoming input signals, the neuron is still able to account for accumulated voltage and generate output pulses. We also do not want to explicitly inject noise through the inputs or make the threshold itself noisy. Instead, randomized control logic is introduced to provide stochasticity in the neuron by randomly adjusting the charge required to fire. This allows a method for introducing stochastic effects in a controlled way.

6.1 Stochastic Neuron Design

Considering existing available designs in the literature for stochastic neurons, we found that there are very few implementations in hardware system [3, 103]. Most stochastic neuron designs are handled with software models based on Gaussian or ReLU activation functions but implementing the complex exponential stochastic functions on a hardware system is really difficult. Some researchers have developed neuron designs with stochasticity using inherent stochasticity of the memristive devices [4, 3]. The drawback here is the reliability of the device considering challenges such as the filament formation. Since, in our system design, we are concentrating on mixed-signal computation (analog inside and digital outside), we propose to design CMOS neurons with added stochasticity. The main reason for this approach is the robustness and the reliability achieved.

The stochastic neuron design introduces stochasticity by forcing the firing rate of the neuron to be probabilistic. A Gaussian distribution is expected in the firing rate depending on the number of incoming input spikes. It is worth noting that the neuron firing rate depends on the charge accumulation rate. Further, charge accumulation is controlled by the membrane capacitance. Thus, the idea here is to occasionally change the membrane capacitance randomly depending on a true random number generator.

To ensure random variations in the membrane capacitance, a chaotic random number generator (RNG) is used. A three-transistor chaotic map circuit (proposed in [27]), along with gating and feedback techniques (discussed in [92]), generate and hold output values at each clock edge. The chaotic map circuit is shown in Fig. 6.1.

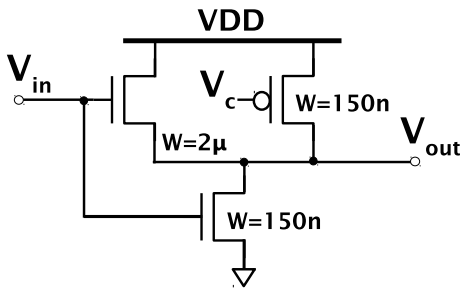

Figure 6.1: Chaotic map circuit from [27].

The initial analog input voltage for the chaotic circuit, V_{seed} is provided by an enable signal *en* before normal operation begins. The bias voltages V_{c1} and V_{c2} in Fig. 6.2 are chosen to ensure that the map circuits operate within a chaotic region. During the firing event, the neuron generates a firing spike and a 3-bit resolution analog-to-digital converter (ADC) captures an analog voltage from the RNG simultaneously. Basically, the ADC helps in splitting the random chaotic voltage from the RNG into three digital control bits: Q_1, Q_2, and Q_3. These digital values are stored in registers until there is a new firing event.

As shown in Fig. 6.3, three capacitors (C_1, C_2, and C_3) are added in parallel to the existing membrane capacitance C_f. Each of the capacitor is connected in series with a pass transistor controlled by one of the output bits from the RNG. These transistors act as switches to "enable" or "disable" the additional capacitors. The value of C_f was lowered slightly from the non-stochastic neuron so that the range of possible capacitance combinations would encapsulate the old value of C_f.

6.1.1 Verifying Stochastic Behavior of Individual Neuron

Since the amount of accumulation due to an incoming spike depends on the membrane capacitance, it was expected that the number of incoming spikes required to surpass the firing threshold would change stochastically along with stochastic variations in the membrane capacitance. This theory has been tested using Cadence Spectre by feeding a periodic 50% duty cycle spike train into the neuron and monitoring the neuron's output spike rate. After acquiring a sufficient number of data points, we plotted the average probability that the neuron will have fired after receiving a given number of input spikes. This curve, shown in Fig. 6.4, convincingly shows the intrinsic stochastic behavior of the neuron.

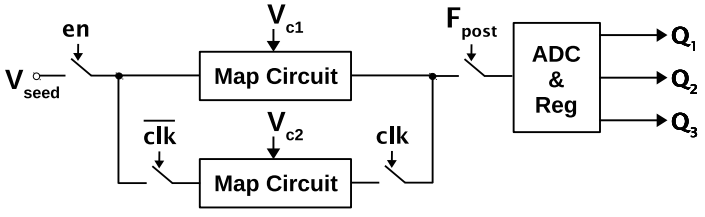

Figure 6.2: Random number generator using scheme from [92].

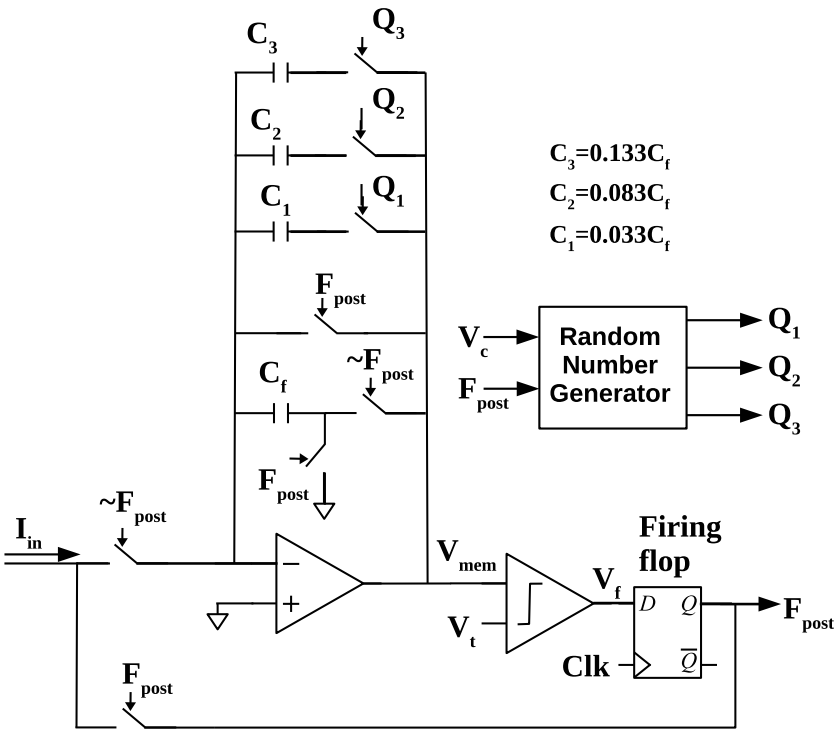

Figure 6.3: Mixed-signal stochastic neuron.

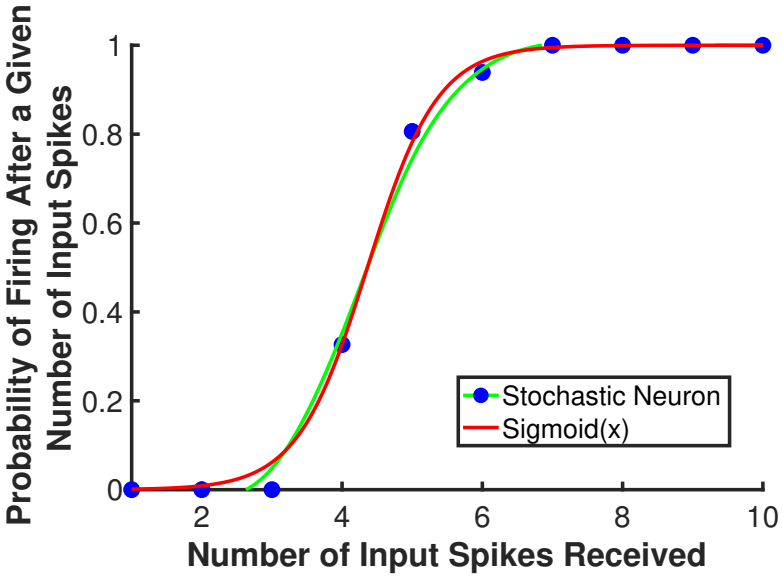

Figure 6.4: Firing distribution for the stochastic IAF neuron compared with a shifted sigmoid function.

The stochastic behavior of the simulated neuron closely approximates a shifted sigmoid, which implies it is an effective implementation of a stochastic binary spiking neuron model. In other words, the neuron will fire an output spike with a probability approximately following a sigmoid distribution.

6.2 Stochasticity Analysis of Neuron

After verifying the stochastic behavior of a single neuron, we moved to the network level. Here, we compare the performance of two identically structured networks, one utilizing deterministically spiking neurons and the other utilizing stochastic spiking neurons. This direct comparison allows us to more easily assess the potential advantages and disadvantages of using stochastic spiking neurons for high-level applications.

A small hand-tooled network structure is used to perform a simple shape recognition task. Specifically, its synaptic weights and delays were designed such that it would be able to recognize triangles and to reject all other shapes with high accuracy. A detailed description of the construction of the network can be found in [81], but for convenience, its topology is shown here in Fig. 6.5.

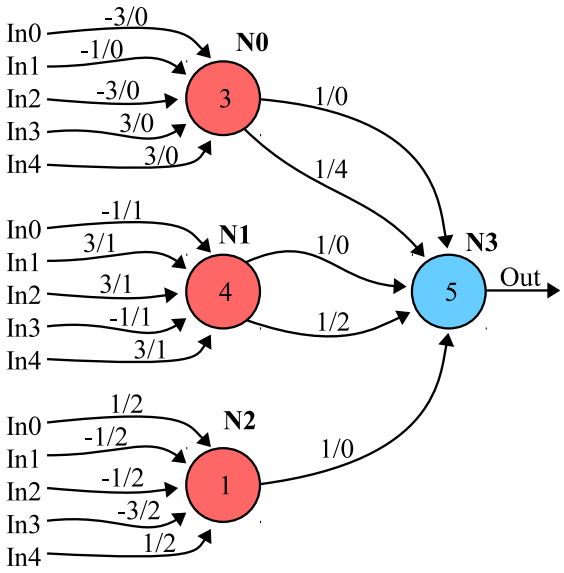

Figure 6.5: Topology of the hand-tooled shape recognition network. The w/d notation refers to the weight/delay of each synapse. The number within each neuron refers to its threshold.

A shape is encoded as a 5×5 array of binary input spikes, where the top row drives input "In0" and the bottom row drives input "In4." Each column in the 5×5 image is given to the network sequentially, ensuring that the shape recognition task becomes a time-series classification problem. The network recognizes a triangle when the output neuron "N3" spikes. To test the stochastic and non-stochastic networks, we constructed datasets using triangles, squares and plus signs. Some of these datasets contained the "ideal" shapes while others contained shapes with added bits of noise. Fig. 6.6 shows some examples of ideal shapes and shapes with up to two added noise bits. The first row represents the "ideal" triangle, square, and plus sign. The second and third rows introduce noise bits. To clarify, the noise added to these shapes was not used in any way to implement stochastic neuron behavior, but simply to make the shapes more difficult to classify.

We simulated the stochastic and non-stochastic networks' responses to the datasets and recorded the resulting recognition accuracies in figures with triangles only, squares only and plus only sets. Fig. 6.7 shows that the stochastic network is significantly better in recognizing noisy triangles than its deterministic counterpart. For example, the stochastic network recognized triangles with 85% accuracy even with 6 noise bits, whereas the non-stochastic network only recognized triangles with 62% accuracy. We believe that the

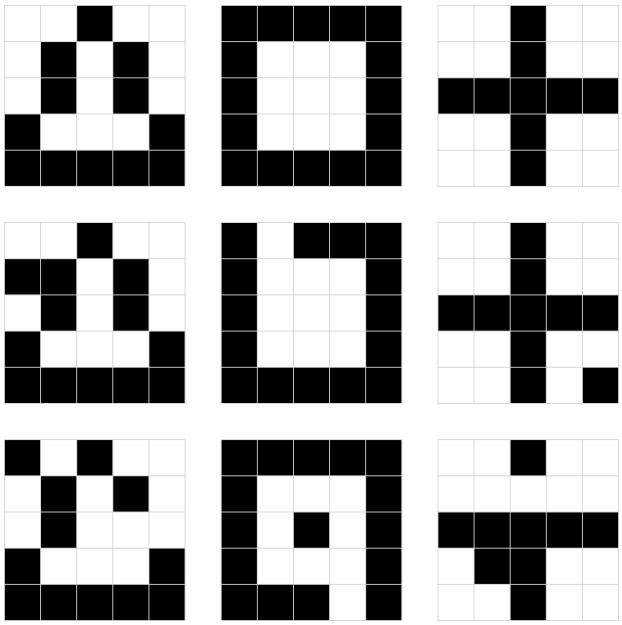

Figure 6.6: 5x5 shapes with and without added noise bits [18].

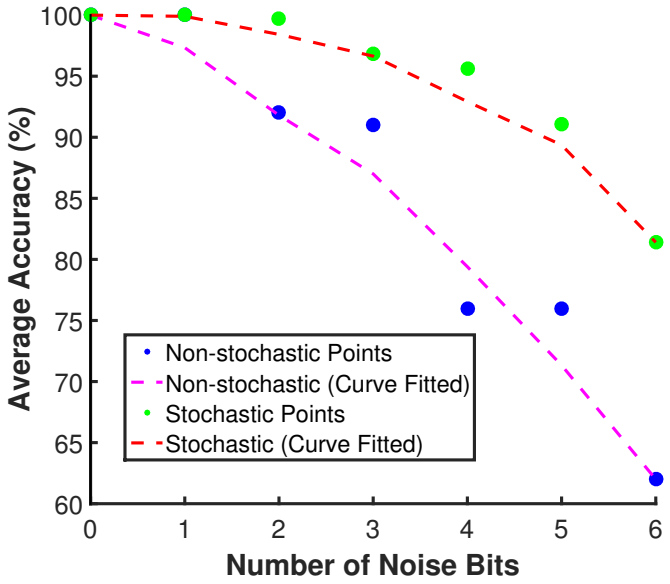

Figure 6.7: Shape recognition non-stochastic vs. stochastic performance on triangle-only set.

stochastic firing rate of the neurons essentially allows the network to perform probabilistic sampling and thus generalize its behavior, accounting more for uncertainty.

In Fig. 6.8 and 6.9, the average recognition accuracy depicts the ability of the stochastic and non-stochastic networks to *reject* squares and plus signs (as they are not triangles). Interestingly, the stochastic network performed less accurately for the dataset of noisy squares in Fig. 6.8. This demonstrates a drawback of the generalizing behavior of the stochastic network. Because it accounted for more uncertainty, it found triangles where they did not actually exist. Since the 5x5 square is already somewhat similar to the 5×5 triangle (due the low resolution), it makes sense that introducing noise bits into a square would create some triangle-like patterns. The stochastic and non-stochastic networks performed similarly for the dat set of plus signs, and we believe it is because there is very little information overlap between the plus sign and the triangle. The ideal plus sign has more pixels near its center, but in general the square and triangle have more pixels around their perimeters. Since the shape types are so different, the networks never encounter situations of high uncertainty, and the generalization behavior of the stochastic neuron does not cause obvious performance differences between them.

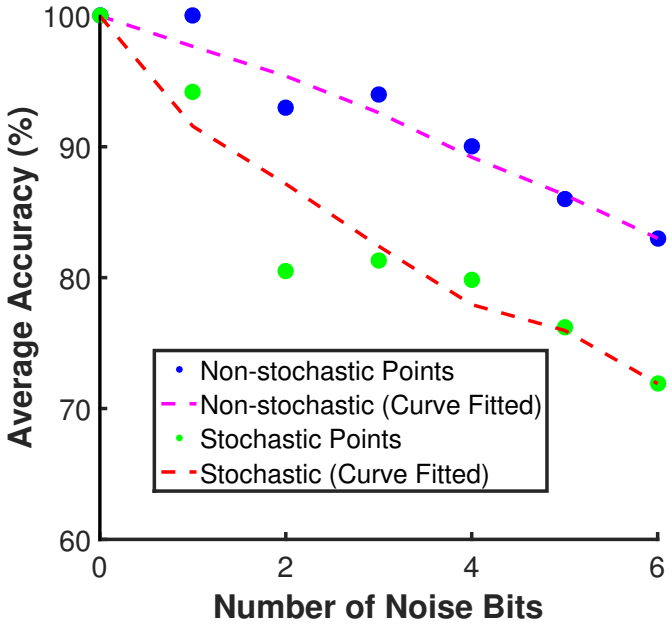

Figure 6.8: Shape Recognition Non-Stochastic vs. Stochastic Performance on Square-Only Set.

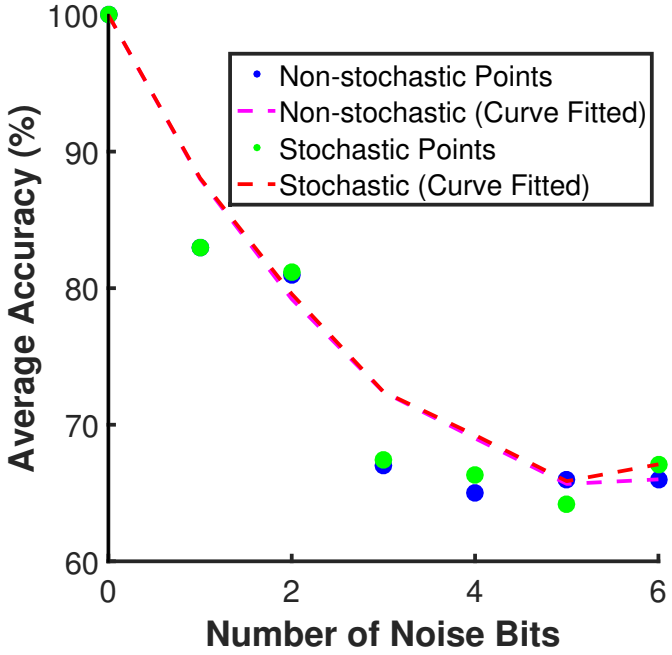

Figure 6.9: Shape Recognition Non-Stochastic vs. Stochastic Performance on Plus-Only Set.

6.3 Power Overhead of Adding Stochastic Dynamics

The proposed stochastic neuron circuit has a similar pattern of energy consumption per spike to the IAF shown in Fig. 3.8. The energy consumption of the stochastic neuron is $9.005pJ$ during the accumulation phase. It is slightly lower than $9.81pJ$ (mentioned in section 3.4), for the non-stochastic neuron. Introducing a lower average value of the membrane capacitance yields both a higher accumulation rate and a lower input current. On the other hand, the firing phase energy consumption, $12.6pJ$, may be marginally higher for the stochastic neuron opposed to the non-stochastic version because of the increased switching activity.

The addition of stochastic feature to the IAF introduces other sources of energy consumption by the RNG and the accompanying ADC and registers. The chaotic oscillator portion of the RNG consumes 191.8 fJ per clock cycle, showing the potential of the 3-transistor map circuit as an energy efficient RNG solution. However, the ADC and registers are likely to be the biggest energy consumers in the proposed circuits. For instance, an available solution based on a 3-bit flash ADC was found to consume approximately 67 nJ per clock cycle. Here, ADC optimization has not been analyzed being outside the scope of this work. However, it is clear from this analysis that ADC selection is a critical design decision for energy efficiency.

To summarize the concept of stochastic neurons, an interesting implementation of introducing stochasticity is discussed in this chapter. Using capacitance variation to add stochastic effect to existing IAF neuron helps in ensuring randomness in firing rate. Thus, there is a generalization behavior introduced to the network because of using stochastic neurons. This behavior helps the networks specifically here, the shape recognition network to gain better accuracy with added noise bits or randomness. The results from this analysis also direct toward the concept of generalizing with input information overlap. This could be useful for networks while working with online learning by updating the synaptic weights based on the generalizing behavior. Moreover, this is a full CMOS approach of introducing stochasticity in the neurons even though there are works that involve emerging devices. Thus, this approach also ensures a controlled and robust way to add stochastic effects to a neuron.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

Neuromorphic computing, being one of the promising alternative computing architectures, is leveraged here to improve computational energy and area efficiency. Neuromorphic computing is also shown to act as an efficient platform for implementing complex neural networks. Since memristors are leveraged as the building blocks for synapses, gains in energy efficiency are ensured at the component level design. If we take a system level perspective, the memristive mixed-signal neuromorphic system follows a synchronous version of NIDA [86] architecture which involves spiking neural networks, more specifically recurrent neural networks. This type of network is commonly used in spatio-temporal classification which often requires complex network topologies. However, the system discussed in this dissertation leverages a genetic algorithm to produce sparse recurrent neural networks which are comparatively smaller than conventional deep neural networks. Hence, gains in energy and area efficiency are also achieved at the system level. Memristive mixed-signal neuromorphic computing is therefore one of the most promising available approaches to move forward the state of the art in area and energy-efficient specialized hardware for Artificial neuromorphic systems. To summarize this work, the following points are listed as highlights:

- A twin memristive synapse with a control block for online learning has been designed. Layouts of the synapse have been completed using Cadence Virtuoso and integrated with peripheral circuits.

- An integrate and fire or IAF neuron with an analog core and digital periphery is designed to ensure better use of digital communication. The neuron layout was also completed using Cadence tools and integrated with synapses in different combinations to verify neuron characteristics for different synaptic weights when integrated with the full system.

- Our synchronous digital long term plasticity or DLTP approach introduces one cycle based learning.

- An algorithm has been established to estimate energy for the neuromorphic system in high-level simulations based on activity factors. These activity factors are captured from the high-level simulator and then used with per spike energy estimation determined from the low-level simulation of key components (synapses and neurons).

- Widely used datasets are used to analyze the effect of online learning in training neural networks for the proposed system and it is proven that the DLTP approach ensures efficiency in power and area with mixed-signal circuit implementations.

- An interesting version of a mixed-signal neuron with stochastic effects has been proposed. This stochastic neuron presents a reliable way to introduce probabilistic interference in neural networks. For the proof of concept, a shape recognition network has been simulated with both regular and stochastic neurons with added noise bits. It is shown in Chapter 6 that, when considering added noise, the probabilistic features in neuron becomes advantageous.

7.2 Future Work

The importance of alternative computing techniques is extensively high in minimizing the energy and performance gap. That is why neuromorphic computing is one of the best

available options. Since this dissertation investigates on an innovative approach with memristive materials and CMOS ICs, there is definitely a wide scope of future work based on this dissertation. Following are some interesting directions for future works that can leverage this work and contribute more to the neuromorphic computing community.

7.2.1 Study of Stochastic Neuron in Advanced Level

A mixed-signal CMOS neuron with stochastic nature has been discussed in Chapter 6. The low-level circuit details have been presented with simulation results and a small shape recognition application. Since the results show promising features while stochasticity is added, there are plenty of directions to further explore this design. Following are some directions for future work regarding the stochastic neuron.

- Exploring the result of using the stochastic neuron in large applications such as classification or spatio-temporal applications so that there are comparisons in using both deterministic and stochastic neurons.

- Study the DLTP mechanism on stochastic neurons. This might be an interesting study because the learning of stochastic neurons might be different than the deterministic neurons because of their probabilistic nature. Also it might build up different learning rates while online learning.

- Study the effects of neural networks with a combination of both stochastic and non-stochastic neurons, as both neuron types have advantages over some tasks. Thus, it would be interesting to analyze the results of combining both.

7.2.2 Leveraging Energy Estimation Algorithm

The energy estimation algorithm discussed in this dissertation has been one of the main contributions of this work. It involves accurate circuit-level energy consumption as well as faster high-level energy estimation calculation. Since, technologies with low energy consumption will thrive in future, this algorithm helps in establishing a connection between

the hardware circuit components and the high-level model. This way, it is easier to design an energy-efficient system.

Multi-objective training has been popular lately because it can be used in several fields of sciences starting from business to engineering. Machine learning has been utilizing this multi-objective training recently because it helps in optimizing different cost-functions and help in establishing optimal networks. The energy estimation algorithm can be leveraged in this type of training. Because it would be interesting to generate networks keeping the energy optimization active since it would help to optimize the network performance while optimizing the energy consumption of the system during training. Hence, it would be advantageous to ensure an energy efficient system.

Bibliography

[1] Aamir, S. A., Müller, P., Hartel, A., Schemmel, J., and Meier, K. (2016). A highly tunable 65-nm cmos lif neuron for a large scale neuromorphic system. In *ESSCIRC Conference 2016: 42nd European Solid-State Circuits Conference*, pages 71–74. IEEE. 9

[2] Adhikari, S. P., Yang, C., Kim, H., and Chua, L. O. (2012). Memristor bridge synapse-based neural network and its learning. *IEEE Transactions on Neural Networks and Learning Systems*, 23(9):1426–1435. 15

[3] Al-Shedivat, M., Naous, R., Cauwenberghs, G., and Salama, K. N. (2015a). Memristors empower spiking neurons with stochasticity. *IEEE journal on Emerging and selected topics in circuits and systems*, 5(2):242–253. 69

[4] Al-Shedivat, M., Naous, R., Neftci, E., Cauwenberghs, G., and Salama, K. N. (2015b). Inherently stochastic spiking neurons for probabilistic neural computation. In *2015 7th International IEEE/EMBS Conference on Neural Engineering (NER)*, pages 356–359. IEEE. 10, 69

[5] Alibart, F., Zamanidoost, E., and Strukov, D. B. (2013). Pattern classification by memristive crossbar circuits using ex situ and in situ training. *Nature communications*, 4. 8, 19

[6] Amer, S., Rose, G. S., Beckmann, K., and Cady, N. C. (2017a). Design techniques for in-field memristor forming circuits. In *Proceedings of IEEE International Midwest Symposium on Circuits and SystemsMWSCAS*, Boston, Massachusetts. 106

[7] Amer, S., Sayyaparaju, S., Rose, G. S., Beckmann, K., and Cady, N. C. (2017b). A practical hafnium-oxide memristor model suitable for circuit design and simulation. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4. 17

[8] Bartolozzi, C., Nikolayeva, O., and Indiveri, G. (2008). Implementing homeostatic plasticity in vlsi networks of spiking neurons. In *2008 15th IEEE International Conference on Electronics, Circuits and Systems*, pages 682–685. IEEE. 12

[9] Belhadj, B., Tomas, J., Bornat, Y., Daouzli, A., Malot, O., and Renaud, S. (2009). Digital mapping of a realistic spike timing plasticity model for real-time neural simulations. In

Proceedings of the XXIV conference on design of circuits and integrated systems, pages 1–6. 63

[10] Berzina, T., Erokhina, S., Camorani, P., Konovalov, O., Erokhin, V., and Fontana, M. (2009). Electrochemical control of the conductivity in an organic memristor: a time-resolved x-ray fluorescence study of ionic drift as a function of the applied voltage. *ACS Applied materials & interfaces*, 1(10):2115–2118. 16

[11] Bi, G. and Poo, M. (2001). Synaptic modification by correlated activity: Hebb's postulate revisited. *Annual review of neuroscience*, 24(1):139–166. 23

[12] Campbell, K. A., Drake, K. T., and Smith, E. H. B. (2016). Pulse shape and timing dependence on the spike-timing dependent plasticity response of ion-conducting memristors as synapses. *Frontiers in Bioengineering and Biotechnology*, 4. 39

[13] Cantley, K. D., Subramaniam, A., Stiegler, H. J., Chapman, R. A., and Vogel, E. M. (2012). Neural learning circuits utilizing nano-crystalline silicon transistors and memristors. *IEEE transactions on neural networks and learning systems*, 23(4):565–573. 16

[14] Cassidy, A., Andreou, A. G., and Georgiou, J. (2011a). A combinational digital logic approach to stdp. In *2011 IEEE International Symposium of Circuits and Systems (ISCAS)*, pages 673–676. 62

[15] Cassidy, A., Andreou, A. G., and Georgiou, J. (2011b). Design of a one million neuron single fpga neuromorphic system for real-time multimodal scene analysis. In *2011 45th Annual Conference on Information Sciences and Systems*, pages 1–6. IEEE. 11

[16] Cassidy, A. S., Georgiou, J., and Andreou, A. G. (2013). Design of silicon brains in the nano-cmos era: Spiking neurons, learning synapses and neural architecture optimization. *Neural Networks*, 45:4–26. 9

[17] Chakma, G., Adnan, M. M., Wyer, A. R., Weiss, R., Schuman, C. D., and Rose, G. S. (2018a). Memristive mixed-signal neuromorphic systems: Energy-efficient learning at the

circuit-level. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 8(1):125–136. xi, xii, xiii, 17, 20, 26, 27, 30, 31, 44, 48, 60, 61, 62

[18] Chakma, G., Sayyaparaju, S., Weiss, R., and Rose, G. S. (2017). A mixed-signal approach to memristive neuromorphic system design. In *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 547–550. IEEE. xiv, 73

[19] Chakma, G., Skuda, N. D., Schuman, C. D., Plank, J. S., Dean, M. E., and Rose, G. S. (2018b). Energy and area efficiency in neuromorphic computing for resource constrained devices. In *Proceedings of the 2018 on Great Lakes Symposium on VLSI*, pages 379–383. ACM. xiii, xiv, 64, 65

[20] Chanthbouala, A., Garcia, V., Cherifi, R. O., Bouzehouane, K., Fusil, S., Moya, X., Xavier, S., Yamada, H., Deranlot, C., Mathur, N. D., et al. (2012). A ferroelectric memristor. *Nature materials*, 11(10):860–864. 16

[21] Chien, C.-H., Liu, S.-C., and Steimer, A. (2018). A neuromorphic vlsi circuit for spike-based random sampling. *IEEE Transactions on Emerging Topics in Computing*, 6(1):135–144. 12

[22] Chua, L. O. (1971). Memristor-the missing circuit element. *IEEE Transactions on Circuit Theory*, 18(5):507–519. 8, 16

[23] Dan, Y. and Poo, M.-m. (2004). Spike timing-dependent plasticity of neural circuits. *Neuron*, 44(1):23–30. 7

[24] Dean, M. E., Chan, J., Daffron, C., Disney, A., Reynolds, J., Rose, G., Plank, J. S., Birdwell, J. D., and Schuman, C. D. (2016). An application development platform for neuromorphic computing. In *Neural Networks (IJCNN), 2016 International Joint Conference on*, pages 1347–1354. IEEE. 46

[25] Dean, M. E., Schuman, C. D., and Birdwell, J. D. (2014). Dynamic adaptive neural network array. In *International Conference on Unconventional Computation and Natural Computation*, pages 129–141. Springer. 13, 45

[26] Diehl, P. U., Neil, D., Binas, J., Cook, M., Liu, S.-C., and Pfeiffer, M. (2015). Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE. 63

[27] Dudek, P. and Juncu, V. (2003). Compact discrete-time chaos generator circuit. *Electronics Letters*, 39(20):1431–1432. xiv, 69

[28] Ebong, I. E. and Mazumder, P. (2012). Cmos and memristor-based neural network design for position detection. *Proceedings of the IEEE*, 100(6):2050–2060. 8

[29] Eryilmaz, S. B., Kuzum, D., Jeyasingh, R., Kim, S., BrightSky, M., Lam, C., and Wong, H.-S. P. (2014). Brain-like associative learning using a nanoscale non-volatile phase change synaptic device array. *Frontiers in neuroscience*, 8:205. 15

[30] Esser, S. K., Appuswamy, R., Merolla, P., Arthur, J. V., and Modha, D. S. (2015). Backpropagation for energy-efficient neuromorphic computing. In *Advances in Neural Information Processing Systems*, pages 1117–1125. 7

[31] Farabet, C., Martini, B., Akselrod, P., Talay, S., LeCun, Y., and Culurciello, E. (2010). Hardware accelerated convolutional neural networks for synthetic vision systems. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 257–260. 63

[32] Furber, S. (2016). Large-scale neuromorphic computing systems. *Journal of neural engineering*, 13(5):051001. 11

[33] Furber, S. B., Galluppi, F., Temple, S., and Plana, L. A. (2014). The spinnaker project. *Proceedings of the IEEE*, 102(5):652–665. 11

[34] Garbin, D., Vianello, E., Bichler, O., Rafhay, Q., Gamrat, C., Ghibaudo, G., DeSalvo, B., and Perniola, L. (2015). Hfo 2-based oxram devices as synapses for convolutional neural networks. *IEEE Transactions on Electron Devices*, 62(8):2494–2501. 7

[35] Gordon, C., Preyer, A., Babalola, K., Butera, R. J., and Hasler, P. (2006). An artificial synapse for interfacing to biological neurons. In *2006 IEEE International Symposium on Circuits and Systems*, pages 4–pp. IEEE. 7

[36] Graf, H., Jackel, L., Howard, R., Straughn, B., Denker, J., Hubbard, W., Tennant, D., Schwartz, D., and Denker, J. S. (1986). Vlsi implementation of a neural network memory with several hundreds of neurons. In *AIP conference proceedings*, volume 151, pages 182–187. AIP. 2

[37] Hodgkin, A. L. and Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500–544. 9

[38] Hsu, J. (2014). Ibm's new brain [news]. *IEEE spectrum*, 51(10):17–19. 11

[39] Hu, M., Chen, Y., Yang, J. J., Wang, Y., and Li, H. (2016). A memristor-based dynamic synapse for spiking neural networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*. 19, 39

[40] Hu, M., Li, H., Chen, Y., Wu, Q., Rose, G. S., and Linderman, R. W. (2014a). Memristor crossbar-based neuromorphic computing system: A case study. *IEEE Trans. Neural Netw. Learning Syst.*, 25(10):1864–1878. 8, 18, 19

[41] Hu, M., Li, H., Chen, Y., Wu, Q., Rose, G. S., and Linderman, R. W. (2014b). Memristor crossbar-based neuromorphic computing system: A case study. *IEEE transactions on neural networks and learning systems*, 25(10):1864–1878. 12

[42] Indiveri, G., Chicca, E., and Douglas, R. (2006). A vlsi array of low-power spiking neurons and bistable synapses with spike-timing dependent plasticity. *IEEE transactions on neural networks*, 17(1):211–221. 7

[43] Indiveri, G., Stefanini, F., and Chicca, E. (2010). Spike-based learning with a generalized integrate and fire silicon neuron. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 1951–1954. IEEE. 10, 28

[44] Izhikevich, E. M. (2003). Simple model of spiking neurons. *IEEE Transactions on neural networks*, 14(6):1569–1572. 9, 29

[45] Jo, S. H., Chang, T., Ebong, I., Bhadviya, B. B., Mazumder, P., and Lu, W. (2010a). Nanoscale memristor device as synapse in neuromorphic systems. *Nano letters*, 10(4):1297–1301. 16

[46] Jo, S. H., Chang, T., Ebong, I., Bhadviya, B. B., Mazumder, P., and Lu, W. (2010b). Nanoscale Memristor Device as Synapse in Neuromorphic Systems. *Nano Letters*, 10(4):1297–1301. 23

[47] Jo, S. H., Kim, K.-H., and Lu, W. (2009). High-density crossbar arrays based on a si memristive system. *Nano letters*, 9(2):870–874. 19

[48] Kataeva, I., Merrikh-Bayat, F., Zamanidoost, E., and Strukov, D. (2015). Efficient training algorithms for neural networks based on memristive crossbar circuits. In *2015 International Joint Conference on Neural Networks, IJCNN*, pages 1–8. 18

[49] Kim, H., Sah, M. P., Yang, C., Roska, T., and Chua, L. O. (2012). Memristor bridge synapses. *Proceedings of the IEEE*, 100(6):2061–2070. 8

[50] Kim, K.-H., Gaba, S., Wheeler, D., Cruz-Albrecht, J. M., Hussain, T., Srinivasa, N., and Lu, W. (2011). A functional hybrid memristor crossbar-array/cmos system for data storage and neuromorphic applications. *Nano letters*, 12(1):389–395. 19

[51] Koickal, T. J., Hamilton, A., Tan, S. L., Covington, J. A., Gardner, J. W., and Pearce, T. C. (2007). Analog vlsi circuit implementation of an adaptive neuromorphic olfaction chip. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 54(1):60–73. 8

[52] Kuzum, D., Jeyasingh, R. G., Lee, B., and Wong, H.-S. P. (2011). Nanoelectronic programmable synapses based on phase change materials for brain-inspired computing. *Nano letters*, 12(5):2179–2186. 15

[53] Lee, H., Chen, Y., Chen, P., Wu, T., Chen, F., Wang, C., Tzeng, P., Tsai, M.-J., and Lien, C. (2010). Low-power and nanosecond switching in robust hafnium oxide resistive memory with a thin ti cap. *IEEE Electron Device Letters*, 31(1):44–46. 16

[54] Li, Y., Zhong, Y., Xu, L., Zhang, J., Xu, X., Sun, H., and Miao, X. (2013). Ultrafast synaptic events in a chalcogenide memristor. *Scientific Reports*, 3:1619. 16

[55] Lichman, M. (2013). UCI machine learning repository. xi, 51, 59

[56] Likharev, K., Mayr, A., Muckra, I., and TÜrel, Ö. (2003). Crossnets: High-performance neuromorphic architectures for cmol circuits. *Annals of the New York Academy of Sciences*, 1006(1):146–163. 12

[57] Linares-Barranco, B., Sanchez-Sinencio, E., Rodriguez-Vazquez, A., and Huertas, J. L. (1993). A cmos analog adaptive bam with on-chip learning and weight refreshing. *IEEE Transactions on Neural networks*, 4(3):445–455. 2

[58] Liu, C., Yan, B., Yang, C., Song, L., Li, Z., Liu, B., Chen, Y., Li, H., Wu, Q., and Jiang, H. (2015). A spiking neuromorphic design with resistive crossbar. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE. 41

[59] Liu, C., Yang, Q., Yan, B., Yang, J., Du, X., Zhu, W., Jiang, H., Wu, Q., Barnell, M., and Li, H. (2016). A memristor crossbar based computing engine optimized for high speed and accuracy. In *VLSI (ISVLSI), 2016 IEEE Computer Society Annual Symposium on*, pages 110–115. IEEE. 31, 41

[60] Maass, W. (1997). Networks of spiking neurons: the third generation of neural network models. *Neural networks*, 10(9):1659–1671. 3

[61] Mahowald, M. (1994). The silicon retina. In *An Analog VLSI System for Stereoscopic Vision*, pages 4–65. Springer. 11

[62] Mead, C. (1989). *Analog VLSI and Neural Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. 9, 11

[63] Mead, C. (1990). Neuromorphic electronic systems. *Proceedings of the IEEE*, 78(10):1629–1636. 28

[64] Medeiros-Ribeiro, G., Perner, F., Carter, R., Abdalla, H., Pickett, M. D., and Williams, R. S. (2011a). Lognormal switching times for titanium dioxide bipolar memristors: origin and resolution. *Nanotechnology*, 22(9):095702. 16

[65] Medeiros-Ribeiro, G., Perner, F., Carter, R., Abdalla, H., Pickett, M. D., and Williams, R. S. (2011b). Lognormal switching times for titanium dioxide bipolar memristors: origin and resolution. *Nanotechnology*, 22(9):095702. 17, 40

[66] Mehonic, A., Cueff, S., Wojdak, M., Hudziak, S., Jambois, O., Labbé, C., Garrido, B., Rizk, R., and Kenyon, A. J. (2012). Resistive switching in silicon suboxide films. *Journal of Applied Physics*, 111(7):074507. 16

[67] Merkel, C. E. and Kudithipudi, D. (2014). A current-mode cmos/memristor hybrid implementation of an extreme learning machine. In *Great Lakes Symposium on VLSI 2014, GLSVLSI '14, Houston, TX, USA - May 21 - 23, 2014*, pages 241–242. 18

[68] Misra, J. and Saha, I. (2010). Artificial neural networks in hardware: A survey of two decades of progress. *Neurocomputing*, 74(1):239–255. 3, 9

[69] Mitchell, J. P., Bruer, G., Dean, M. E., Plank, J. S., Rose, G. S., and Schuman, C. D. (2017). Neon: Neuromorphic control for autonomous robotic navigation. In *2017 IEEE International Symposium on Robotics and Intelligent Sensors (IRIS)*, pages 136–142. 64

[70] Mitra, S., Fusi, S., and Indiveri, G. (2009). Real-time classification of complex patterns using spike-based learning in neuromorphic vlsi. *IEEE Transactions on Biomedical Circuits and Systems*, 3(1):32–42. 28

[71] Nishitani, Y., Kaneko, Y., Ueda, M., Morie, T., and Fujii, E. (2012). Three-terminal ferroelectric synapse device with concurrent learning function for artificial neural networks. *Journal of Applied Physics*, 111(12):124108. 15

[72] Noack, M., Krause, M., Mayr, C., Partzsch, J., and Schüffny, R. (2014). Vlsi implementation of a conductance-based multi-synapse using switched-capacitor circuits. In *2014 IEEE international symposium on circuits and systems (ISCAS)*, pages 850–853. IEEE. 7

[73] Oblea, A. S., Timilsina, A., Moore, D., and Campbell, K. A. (2010). Silver chalcogenide based memristor devices. In *Neural Networks (IJCNN), The 2010 International Joint Conference on*, pages 1–3. IEEE. 16

[74] Oster, M. and Liu, S.-C. (2004). A winner-take-all spiking network with spiking inputs. In *Proceedings of the 2004 11th IEEE International Conference on Electronics, Circuits and Systems, 2004. ICECS 2004.*, pages 203–206. IEEE. 7

[75] Panchula, A. (2007). Oscillating-field assisted spin torque switching of a magnetic tunnel junction memory element. US Patent 7,224,601. 16

[76] Pantazi, A., Woźniak, S., Tuma, T., and Eleftheriou, E. (2016). All-memristive neuromorphic computing with level-tuned neurons. *Nanotechnology*, 27(35):355205. 10

[77] Parisien, C., Anderson, C. H., and Eliasmith, C. (2008). Solving the problem of negative synaptic weights in cortical models. *Neural computation*, 20(6):1473–1494. 19

[78] Plesser, H. E. and Gerstner, W. (2000). Noise in integrate-and-fire neurons: From stochastic input to escape rates. *Neural computation*, 12(2):367–384. 68

[79] Ramakrishnan, S., Hasler, P. E., and Gordon, C. (2011). Floating gate synapses with spike-time-dependent plasticity. *IEEE Transactions on Biomedical Circuits and Systems*, 5(3):244–252. 8, 15

[80] Rose, G. S., Manem, H., Rajendran, J., Karri, R., and Pino, R. (2012). Leveraging memristive systems in the construction of digital logic circuits. *Proceedings of the IEEE*, 100(6):2033–2049. 18

[81] Sayyaparaju, S., Weiss, R., and Rose, G. S. (2018). A mixed-mode neuron with on-chip tunability for generic use in memristive neuromorphic systems. In *Proceedings of IEEE Computer Society Annual Symposium on ISVLSI*, Hong Kong, China. 72

[82] Schemmel, J., Fieres, J., and Meier, K. (2008). Wafer-scale integration of analog neural networks. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pages 431–438. IEEE. 11

[83] Schneider, C. and Card, H. (1991a). Analog cmos synaptic learning circuits adapted from invertebrate biology. *IEEE transactions on circuits and systems*, 38(12):1430–1438. 2

[84] Schneider, C. and Card, H. (1991b). Cmos implementation of analog hebbian synaptic learning circuits. In *IJCNN-91-Seattle International Joint Conference on Neural Networks*, volume i, pages 437–442 vol.1. 2

[85] Schuman, C., Birdwell, J., and Dean, M. (2014a). Neuroscience-inspired inspired dynamic architectures. In *Biomedical Science and Engineering Center Conference (BSEC), 2014 Annual Oak Ridge National Laboratory*, pages 1–4. 4, 12, 44

[86] Schuman, C. D., Birdwell, J. D., and Dean, M. E. (2014b). Spatiotemporal classification using neuroscience-inspired dynamic architectures. *Procedia Computer Science*, 41:89–97. 5th Annual International Conference on Biologically Inspired Cognitive Architectures, 2014 BICA. 3, 77

[87] Schuman, C. D., Plank, J. S., Disney, A., and Reynolds, J. (2016). An evolutionary optimization framework for neural networks and neuromorphic architectures. In *Neural Networks (IJCNN), 2016 International Joint Conference on*, pages 145–154. IEEE. 45, 46

[88] Schuman, C. D., Plank, J. S., Rose, G. S., Chakma, G., Wyer, A., Bruer, G., and Laanait, N. (2017). A programming framework for neuromorphic systems with emerging technologies. In *Proceedings of the 4th ACM International Conference on Nanoscale Computing and Communication*, page 15. ACM. 45

[89] Seo, J.-s., Brezzo, B., Liu, Y., Parker, B. D., Esser, S. K., Montoye, R. K., Rajendran, B., Tierno, J. A., Chang, L., Modha, D. S., et al. (2011). A 45nm cmos neuromorphic chip with a scalable architecture for learning in networks of spiking neurons. In *Custom Integrated Circuits Conference (CICC), 2011 IEEE*, pages 1–4. IEEE. 2

[90] Serrano-Gotarredona, T., Masquelier, T., Prodromakis, T., Indiveri, G., and Linares-Barranco, B. (2013a). Stdp and stdp variations with memristors for spiking neuromorphic learning systems. *Frontiers in neuroscience*, 7:2. 12

[91] Serrano-Gotarredona, T., Masquelier, T., Prodromakis, T., Indiveri, G., and Linares-Barranco, B. (2013b). Stdp and stdp variations with memristors for spiking neuromorphic learning systems. *Frontiers in Neuroscience*, 7(2). 23

[92] Shanta, A. S., Majumder, M. B., Hasan, M. S., Uddin, M., and Rose, G. S. (2018). Design of a reconfigurable chaos gate with enhanced functionality space in 65nm cmos. In *2018 IEEE 61st International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 1016–1019. IEEE. xiv, 69, 70

[93] Shukla, A., Kumar, V., and Ganguly, U. (2017). A software-equivalent snn hardware using rram-array for asynchronous real-time learning. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 4657–4664. 62

[94] Skuda, N. D., Schuman, C. D., Chakma, G., Plank, J. S., and Rose, G. S. (2018). High-level simulation for spiking neuromorphic computing systems. In *IEEE International Symposium on Circuits and Systems (ISCAS)*. xiii, 52, 53

[95] Snider, G. S. (2008). Spike-timing-dependent learning in memristive nanodevices. In *IEEE International Symposium on Nanoscale Architectures, 2008 (NANOARCH)*, pages 85–92. 23

[96] Srinivasan, V., Dugger, J., and Hasler, P. (2005). An adaptive analog synapse circuit that implements the least-mean-square learning rule. In *2005 IEEE International Symposium on Circuits and Systems*, pages 4441–4444. IEEE. 7

[97] Strukov, D. B., Likharev, K. K., and Waser, R. (2012). Reconfigurable nano-crossbar architectures. *Nanoelectronics and information technology*, pages 543–562. 12

[98] Suri, M., Bichler, O., Querlioz, D., Traoré, B., Cueto, O., Perniola, L., Sousa, V., Vuillaume, D., Gamrat, C., and DeSalvo, B. (2012). Physical aspects of low power synapses based on phase change memory devices. *Journal of Applied Physics*, 112(5):054904. 15

[99] Terwilliger, A. M., Perdue, G. N., Isele, D., Patton, R. M., and Young, S. R. (2017). Vertex reconstruction of neutrino interactions using deep learning. In *Neural Networks (IJCNN), 2017 International Joint Conference on*, pages 2275–2281. IEEE. xiv, 66, 67

[100] Tovar, G. M., Fukuda, E. S., Asai, T., Hirose, T., and Amemiya, Y. (2007). Analog cmos circuits implementing neural segmentation model based on symmetric stdp learning. In *International Conference on Neural Information Processing*, pages 117–126. Springer. 8

[101] Truong, S. N., Shin, S., Byeon, S.-D., Song, J., and Min, K.-S. (2015). New twin crossbar architecture of binary memristors for low-power image recognition with discrete cosine transform. *IEEE Transactions on Nanotechnology*, 14(6):1104–1111. 19

[102] Truong, S. N., Van Pham, K., Yang, W., Jo, A., Lee, M. J., Mo, H.-S., and Min, K.-S. (2017). Time-shared twin memristor crossbar reducing the number of arrays by half for pattern recognition. *Nanoscale research letters*, 12(1):205. 19

[103] Tuma, T., Pantazi, A., Le Gallo, M., Sebastian, A., and Eleftheriou, E. (2016). Stochastic phase-change neurons. *Nature nanotechnology*, 11(8):693. 10, 69

[104] Uddin, M., Majumder, M. B., Rose, G. S., Beckmann, K., Manem, H., Alamgir, Z., and Cady, N. C. (2016). Techniques for improved reliability in memristive crossbar puf circuits. In *VLSI (ISVLSI), 2016 IEEE Computer Society Annual Symposium on*, pages 212–217. IEEE. 17, 40

[105] Wang, H., Li, H., and Pino, R. E. (2012). Memristor-based synapse design and training scheme for neuromorphic computing architecture. In *The 2012 International Joint Conference on Neural Networks (IJCNN)*, pages 1–5. IEEE. 12

[106] Wang, H., Zhao, Q., Ni, Z., Li, Q., Liu, H., Yang, Y., Wang, L., Ran, Y., Guo, Y., Hu, W., et al. (2018). A ferroelectric/electrochemical modulated organic synapse for ultraflexible, artificial visual-perception system. *Advanced Materials*, 30(46):1803961. 9, 15

[107] Wang, L., Lu, S.-R., and Wen, J. (2017a). Recent advances on neuromorphic systems using phase-change materials. *Nanoscale research letters*, 12(1):347. 9, 15

[108] Wang, Z., Joshi, S., Savelev, S. E., Jiang, H., Midya, R., Lin, P., Hu, M., Ge, N., Strachan, J. P., Li, Z., et al. (2017b). Memristors with diffusive dynamics as synaptic emulators for neuromorphic computing. *Nature materials*, 16(1):101. 15

[109] Wright, C. D., Hosseini, P., and Diosdado, J. A. V. (2013). Beyond von-neumann computing with nanoscale phase-change memory devices. *Advanced Functional Materials*, 23(18):2248–2254. 10

[110] Wu, X., Saxena, V., and Campbell, K. A. (2014). Energy-efficient STDP-based learning circuits with memristor synapses. In *Proceedings of SPIE*, volume 9119, pages 911906–1–911906–7. 10, 29, 41

[111] Wu, X., Saxena, V., and Zhu, K. (2015). A cmos spiking neuron for dense memristor-synapse connectivity for brain-inspired computing. In *Neural Networks (IJCNN), 2015 International Joint Conference on*, pages 1–6. IEEE. 22

[112] Xia, Q., Robinett, W., Cumbie, M. W., Banerjee, N., Cardinali, T. J., Yang, J. J., Wu, W., Li, X., Tong, W. M., Strukov, D. B., et al. (2009). Memristor- cmos hybrid integrated circuits for reconfigurable logic. *Nano letters*, 9(10):3640–3645. 19

[113] Yang, J. J., Pickett, M. D., Li, X., Ohlberg, D. A., Stewart, D. R., and Williams, R. S. (2008). Memristive switching mechanism for metal/oxide/metal nanodevices. *Nature nanotechnology*, 3(7):429. 8

[114] Yang, J. J., Zhang, M., Strachan, J. P., Miao, F., Pickett, M. D., Kelley, R. D., Medeiros-Ribeiro, G., and Williams, R. S. (2010). High switching endurance in taox memristive devices. *Applied Physics Letters*, 97(23):232102. 16, 17, 40

[115] Yepes, A. J., Tang, J., and Mashford, B. S. (2017). Improving classification accuracy of feedforward neural networks for spiking neuromorphic chips. *arXiv preprint arXiv:1705.07755*. 63

[116] Zhu, X., Yang, X., Wu, C., Wu, J., and Yi, X. (2013). Hamming network circuits based on cmos/memristor hybrid design. *IEICE Electronics Express*, pages 10–20130404. 19

[117] Ziegler, M. and Kohlstedt, H. (2013). Mimic synaptic behavior with a single floating gate transistor: A memflash synapse. *Journal of Applied Physics*, 114(19):194506. 8, 15

Appendices

Appendix A

VerilogA Code for IAF Neuron Design

A verilog-a code is written for the IAF neuron design to verify the behavior of the real neuron. This neuron model follows the functionalities of the circuit-level mixed-signal IAF neuron meaning it takes input currents from the connected pre-synaptic inputs and delivers an output as a voltage pulse. It also presents the feature of refractory period for firing phase and has a capacitance value for accumulation phase.

```
// VerilogA for MemrDANNA_10lpe_v2, neu_combo_v2_verilog, veriloga

`include "constants.vams"
`include "disciplines.vams"

module neu_combo_v2_verilog (in_pulse1, in_pulse2, in_pulse3,
    in_pulse4, in_pulse5, final_fire, feedback, vo, neu_in, vth,
    vss, vdd, clk, comparator_sig, accu);
input vth, vss, vdd, clk, in_pulse1, in_pulse2, in_pulse3,
    in_pulse4, in_pulse5;
inout neu_in, feedback, comparator_sig;
output final_fire, vo, accu;
```

```
electrical neu_in, vth, vss, vdd, clk,final_fire, vo, feedback,
   feedback_b, final_fire_b, accu, comparator_sig, fire_b,
   in_pulse1, in_pulse2, in_pulse3, in_pulse4, in_pulse5;

parameter real cap = 60p from (0:inf);
parameter real td = 0 from [0:inf);      // delay from clock to q
parameter real tt = 0 from [0:inf);      // transition time of
   output signals
parameter integer dir = +1 from [-1:+1] exclude 0;

real x ,temp, x1, temp1, nowtime, pasttime, state1, state2, z;

analog begin
        // integrator
        @(initial_step) begin
                x = 0;
                temp = 0;
                temp1 = 0;
                x1 = 0;
                nowtime = 0;
                pasttime = 0;
        end
        nowtime = $abstime;
        x = I(neu_in);
        if (abs(x-x1)>50p)
                temp = (x-x1)*(nowtime-pasttime)+temp;
        else
                temp = x*(nowtime-pasttime)+temp;
        pasttime = $abstime;
        x1=I(neu_in);
```

```
        if (V(feedback)>=0) begin
                temp = 0;
                x = 0;
        end
        V(accu) <+ temp*(-1/cap);

// comparator
        if (V(accu) < V(vth))
                V(comparator_sig) <+ V(vdd);
        else
                V(comparator_sig) <+ V(vss);
//dff 1
    @(cross(V(clk) -V(vdd)/2, dir))
                state1 = (V(comparator_sig) > V(vdd)/2);

    V(feedback) <+ transition( state1 ? V(vdd) : V(vss), td, tt );
        V(feedback_b) <+ transition( state1 ? V(vss) : V(vdd), td,
            tt );
// transition gate1
        if ((V(feedback) == V(vdd)) && V(clk)==V(vdd))  //high
           state set to this value
                V(neu_in) <+ V(vdd);
        else    //low state set to this value
                V(neu_in) <+ 0;
// dff 2
    @(cross(V(clk) -V(vdd)/2, dir))
                state2 = (V(feedback) > V(vdd)/2);
    V(final_fire) <+ transition( state2 ? V(vdd) : V(vss), td, tt
        );
```

```
        V(final_fire_b) <+ transition( state2 ? V(vss) : V(vdd),
            td, tt );
// transition gate2
        if ((V(final_fire) == V(vdd)) && V(clk)==V(vdd))          //
            high state set to this value
                V(neu_in) <+ V(vss);
        else     //low state set to this value
                V(neu_in) <+ 0;
// half wave
        z = V(final_fire);
        if (V(final_fire)<=0)
                z=0;
        V(vo) <+ z;
// nor fre_t and fre
        if (V(final_fire) < V(vdd) && V(feedback) < V(vdd))
                V(fire_b) <+ V(vdd);
        else
                V(fire_b) <+ V(vss);
// or gate
        if (V(final_fire) < V(vdd) && V(feedback) < V(vdd))
                V(fire_b) <+ V(vdd);
        else
                V(fire_b) <+ V(vss);

end
endmodule
```

Appendix B

Testing Strategy for Test-structures

A test chip has been taped out for the proposed mrDANNA design. Besides, a handful of test structures have been taped out as well. To test the mixed-signal neuron design different test structures have been developed and the strategies to test the fabricated test structures have been determined. For now, the fabrication steps have upto three metal layers and it is not possible to test the test structures with probe station since we need upto BA layer to reach the measurement node.

B.1 Test structure with Resistor (single and multiple) and Mixed-signal Neuron

This test structure consists of two small tests. One with a single resistor and the other with two resistors to implement synapse. These two tests are included in the same 12x2 probe pad to utilize the pins. The pin arrangement of the probe pad is shown in Fig. B.1 where the upper pins are noted from t1 to t12 and the bottom pins are noted from b1 to b12.

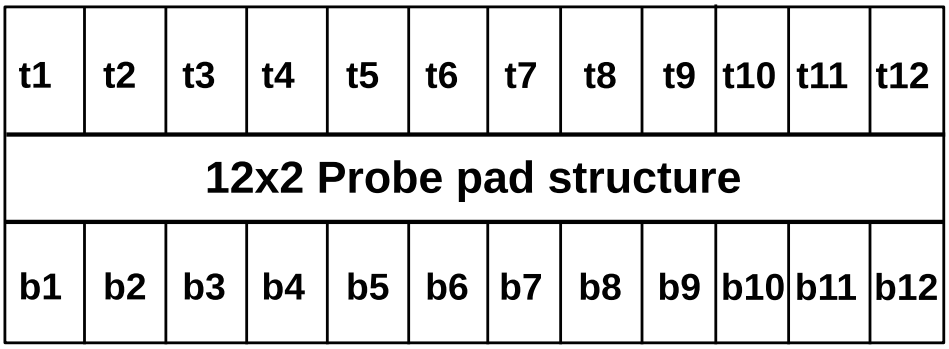

Figure B.1: 12x2 probe pad structure.

For this particular test structure, pins are assigned following the stated arrangement shown in table B.1. There are two sets of power sources of VDD and VSS for two different tests. The test with single resistor is built to verify the accumulation of the mixed-signal neuron. The thresholds can be varied with different DC input voltage. The resistance value is fixed for this test. The single resistance value here represents the synaptic weight as well as conductance. table B.2 shows the signals to different pins for this test, Fig. B.2 shows the schematic and Fig. B.3 shows the input and output signals of the test structure. On the other hand, the test with twin resistor is a prototype of the twin memristor explained in chapter 3. Here the resistive synapse represents maximum effective conductance possible with $9K$ and $15K$ resistances. table B.3 represents the signal connection to the pins, Fig. B.4 shows the schematic and Fig. B.5 shows the input and output signals for this test structure.

Table B.1: Pin assignment of test structure B.1.

Pin name	Pin Connection	Pin Direction	Pin name	Pin Connection	Pin Direction
t1	Not Connected	floating	b1	Not Connected	floating
t2	fre1	output	b2	Not Connected	floating
t3	Fire1	output	b3	Vth1	input
t4	CLK1	input	b4	GND1	input
t5	Vb1	input	b5	Vref1	input
t6	VSS	input	b6	SFire1	input
t7	VDD1	input	b7	fre	output
t8	Fire	output	b8	Vth	input
t9	CLK	input	b9	GND	input
t10	Vb	input	b10	Vref	input
t11	SFire	input	b11	SFireb	input
t12	VDD	input	b12	VSS	input

Table B.2: Signal description of single resistor test structure in B.1.

Pin Attribute	Pin type	Signal Description
VDD1	DC input	Power source for single resistor test (1.2V)
VSS1	DC input	Power source for single resistor test (0V)
SFire1	DC pulse-train input	Input voltage pulse for single resistor test
Vref1	DC input	Reference voltage for single resistor test (0.6V)
Vb1	DC input	Reference voltage for single resistor test (0.6V)
CLK1	DC pulse-train input	Clock signal Reference voltage for single resistor test (20MHz)
Vth1	DC input	Threshold voltage for single resistor test (multiples of 20mV)
fre1	DC output	Initial output fire for single resistor test
Fire1	DC output	Final output fire for single resistor test

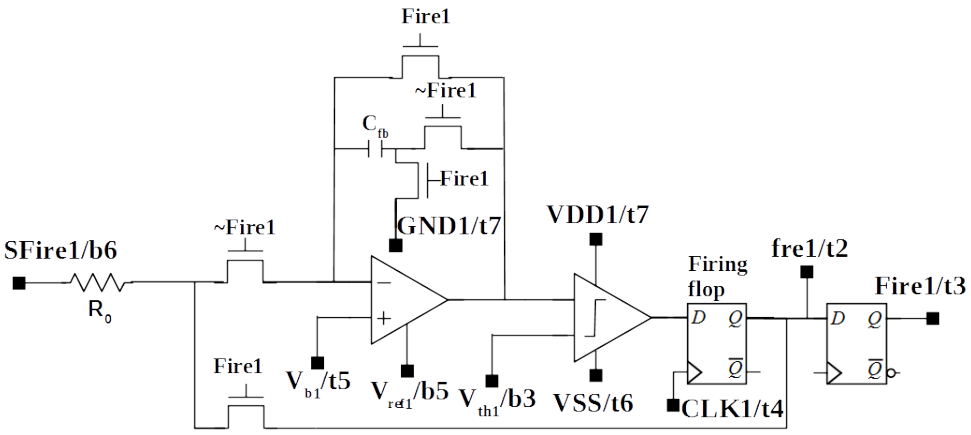

Figure B.2: Schematic for single resistor and single neuron test structure.

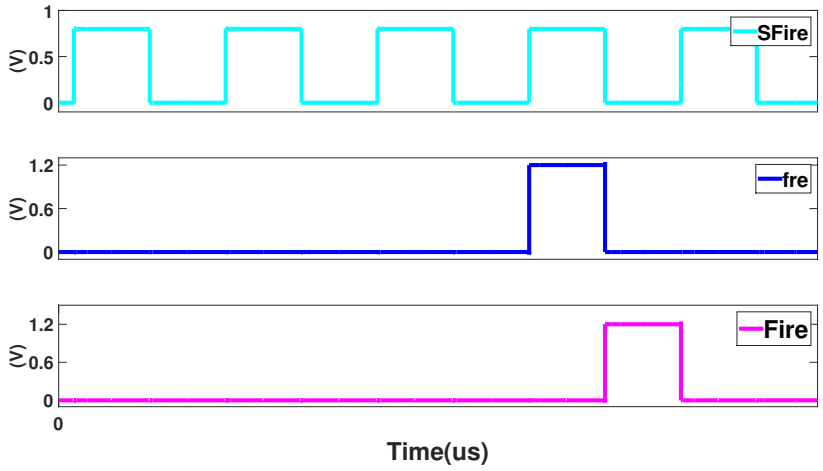

Figure B.3: Simulation of single resistor and single neuron test structure B.1.

Table B.3: Signal description of twin resistor test structure in B.1.

Pin Attribute	Pin type	Signal Description
VDD	DC input	Power source for twin resistor test (1.2V)
VSS	DC input	Power source for twin resistor test (0V)
SFire	DC pulse-train input	Input voltage pulse for twin resistor test
SFireb	DC pulse-train input	Inverted SFire for twin resistor test
Vref	DC input	Reference voltage for twin resistor test (0.6V)
Vb	DC input	Reference voltage for twin resistor test (0.6V)
CLK	DC pulse-train input	Clock signal Reference voltage for twin resistor test (20MHz)
Vth	DC input	Threshold voltage for twin resistor test (multiples of 20mV)
fre	DC output	Initial output fire for twin resistor test
Fire	DC output	Final output fire for twin resistor test

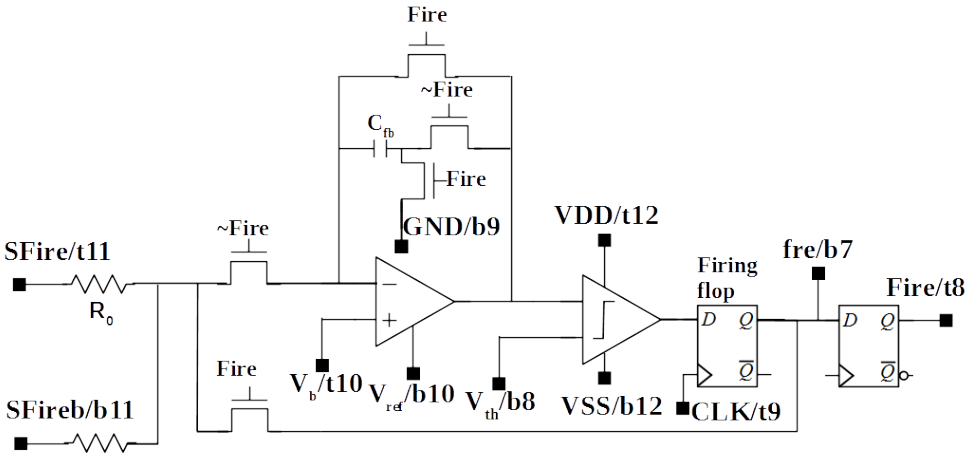

Figure B.4: Schematic for multiple resistor and single neuron test structure B.1.

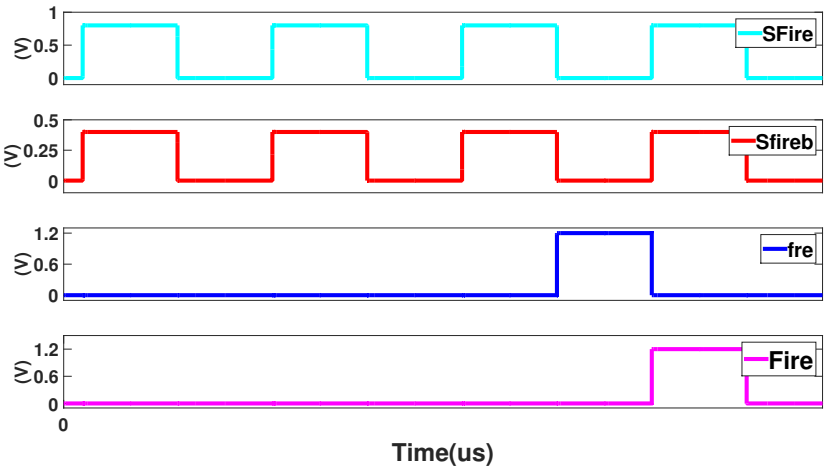

Figure B.5: Simulation of multiple resistor and single neuron test structure B.1.

B.2 Test Structure with Memristive Synapse with Forming Circuit and Mixed-signal Neuron

This test structure shows the connection of a synapse with a mixed-signal neuron. The twin memristive synapse needs to be programmed to a state before we can use it as a synapse device. So, this test structure also includes the forming and programming circuit discussed in [6]. This test structure has the similar pin assignments as B.1. In fact there are additional pins such as $Vformp$ and $Vformn$ for enabling the forming circuits; $Vprogp$ and $Vprogn$ for enabling the programming circuits and $Vpin$ for the programming sequence. The pin assignment details are mentioned in table B.4 and B.5. Fig. B.6 is the schematic of this test structure and Fig. B.7-B.8 shows the input-output signals of the test structure.

Table B.4: Pin assignment of test structure B.2.

Pin name	Pin Connection	Pin Direction	Pin name	Pin Connection	Pin Direction
t1	Not Connected	floating	b1	Not Connected	floating
t2	Not Connected	floating	b2	Not Connected	floating
t3	Not Connected	floating	b3	Not Connected	floating
t4	Vpin	input	b4	Not Connected	floating
t5	Vformp	input	b5	Vprogn	input
t6	Vprogp	input	b6	Vformn	input
t7	VDDH	input	b7	fre	output
t8	Fire	output	b8	Vth	input
t9	CLK	input	b9	GND	input
t10	SFire	input	b10	SFireb	input
t11	Vref	input	b11	Vb	input
t12	VDD	input	b12	VSS	input

Table B.5: Signal description of twin resistor test structure in B.2.

Pin Attribute	Pin type	Signal Description
VDD	DC input	Power source for twin resistor test (1.2V)
VSS	DC input	Power source for twin resistor test (0V)
SFire	DC pulse-train input	Input voltage pulse for twin resistor test
SFireb	DC pulse-train input	Inverted SFire for twin resistor test
Vref	DC input	Reference voltage for twin resistor test (0.6V)
Vb	DC input	Reference voltage for twin resistor test (0.6V)
CLK	DC pulse-train input	Clock signal Reference voltage for twin resistor test (20MHz)
Vth	DC input	Threshold voltage for twin resistor test (multiples of 20mV)
fre	DC output	Initial output fire for twin resistor test
Fire	DC output	Final output fire for twin resistor test
VDDH	DC input	Power source for dgxfets (3.3V)
Vformp	DC input	Forming enabler for positive memristor (3.3V)
Vformn	DC input	Forming enabler for negative memristor (3.3V)
Vprogp	DC input	Programming enabler for positive memristor (3.3V)
Vprogn	DC input	Programming enabler for negative memristor (3.3V)
Vpin	DC pulse-train input	Programming Pin for the synapses (3.3V)

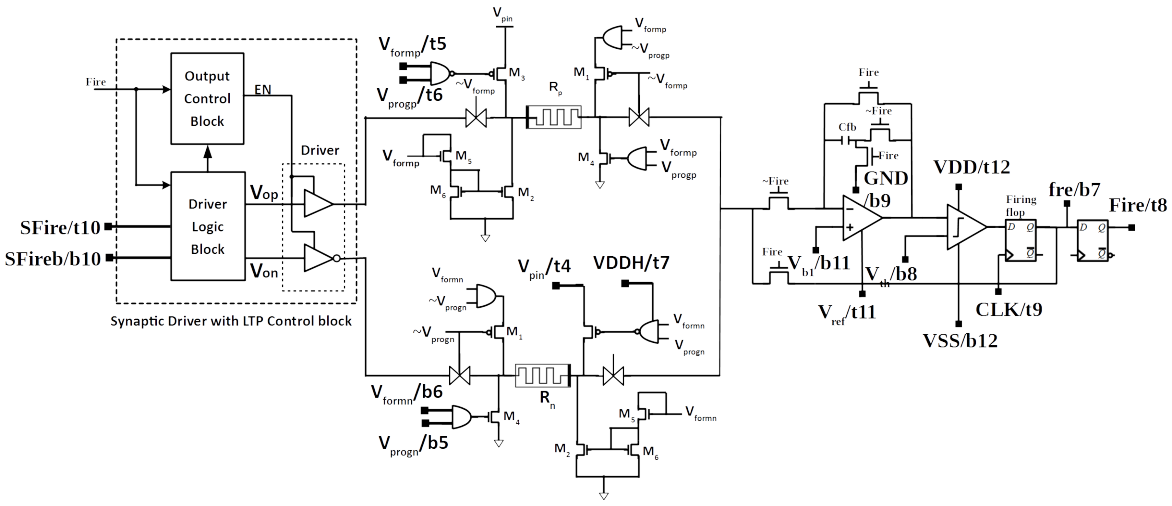

Figure B.6: Schematic for single memristive synapse and single neuron test structure B.2.

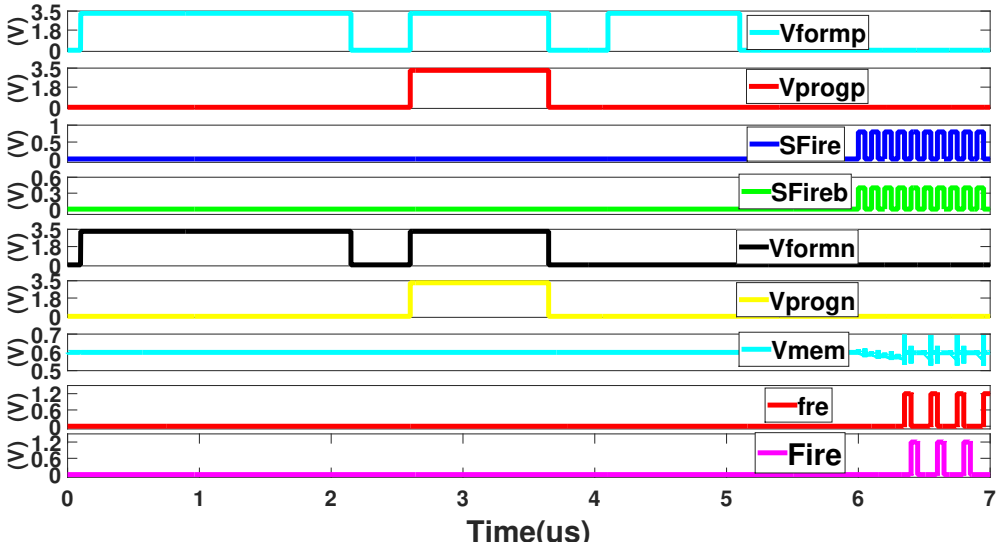

Figure B.7: Simulation of single memristive synapse and single neuron test structure B.2.

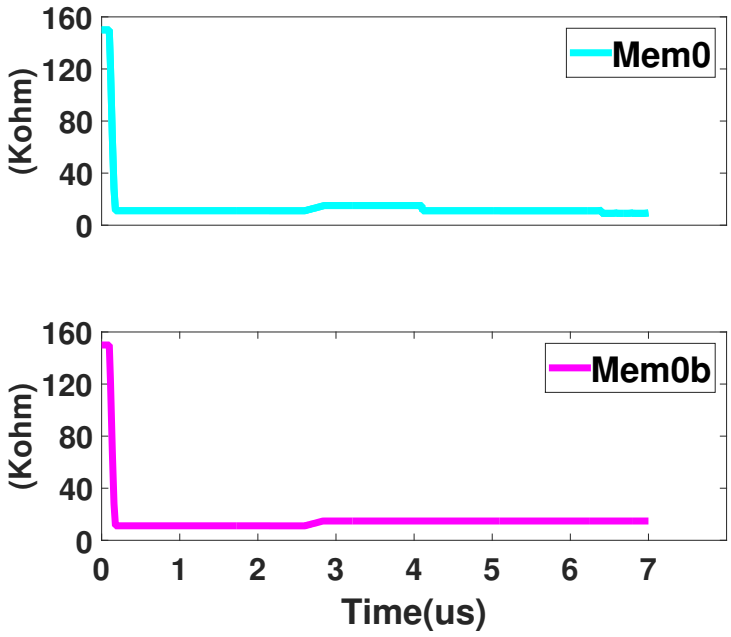

Figure B.8: Simulation of memristor forming.

B.3 Test Structure of a System Prototype

A test structure has been built to verify the prototype of the whole system meaning several (eight here particularly) synapses connected to a mixed signal neuron. This test is designed to verify the mechanism of a memristive neuromprphic core or system. Here, we have used resistive synapses instead of memristive synapse. The motivation behind this test structure was to prove that the memristive core works well with multiple synapse and neuron design. Moreover, the resistive synapses were designed to hold the maximum synaptic weight possible when designed with memristors. So, this test structure can be shown as a prototype of the designed core.

Like other test structures, this one includes the power sources VDD and VSS; some DC reference voltages for the neuron such as $Vref$, Vb, GND and Vth; a clock, CLK for the system, $eight$ input pulse trains from $SFire - SFire7$ for synaptic inputs and output signals, $Fire$ and fre. All of these pins are described in table B.6 and B.7. Fig. B.9 is the schematic for this test structure. The simulation result of this structure is shown in Fig.

B.10. Here eight synaptic inputs are supplied to the corresponding synapses. The volatge accumulation is shown with signal $Vmem$ and the threshold voltage is set to $500mV$ here. When the accumulated voltage crosses the Vth, fre signal indicates the generation of output $Fire$.

Table B.6: Pin assignment of test structure B.3.

Pin name	Pin Connection	Pin Direction	Pin name	Pin Connection	Pin Direction
t1	Not Connected	floating	b1	Not Connected	floating
t2	Not Connected	floating	b2	Not Connected	floating
t3	Vth	input	b3	Not Connected	floating
t4	SFire7	input	b4	GND	input
t5	SFire5	input	b5	SFire6	input
t6	SFire3	input	b6	SFire4	input
t7	SFire1	input	b7	SFire2	input
t8	Vb	input	b8	SFire	input
t9	V800	input	b9	V400	input
t10	Fire	output	b10	fre	output
t11	CLK	input	b11	Vref	input
t12	VDD	input	b12	VSS	input

Table B.7: Signal description of twin resistor test structure in B.3.

Pin Attribute	Pin type	Signal Description
VDD	DC input	Power source (1.2V)
VSS	DC input	Power source (0V)
SFire	DC pulse-train input	Input voltage pulse for synapse0
SFire1	DC pulse-train input	Input voltage pulse for synapse1
SFire2	DC pulse-train input	Input voltage pulse for synapse2
SFire3	DC pulse-train input	Input voltage pulse for synapse3
SFire4	DC pulse-train input	Input voltage pulse for synapse4
SFire5	DC pulse-train input	Input voltage pulse for synapse5
SFire6	DC pulse-train input	Input voltage pulse for synapse6
SFire7	DC pulse-train input	Input voltage pulse for synapse7
CLK	DC pulse-train input	Clock signal Reference voltage for twin resistor test (20MHz)
Vb	DC input	Reference voltage for twin resistor test (0.6V)
Vref	DC input	Reference voltage for twin resistor test (0.6V)
Vth	DC input	Threshold voltage for twin resistor test (multiples of 20mV)
fre	DC output	Initial output fire for twin resistor test
Fire	DC output	Final output fire for twin resistor test

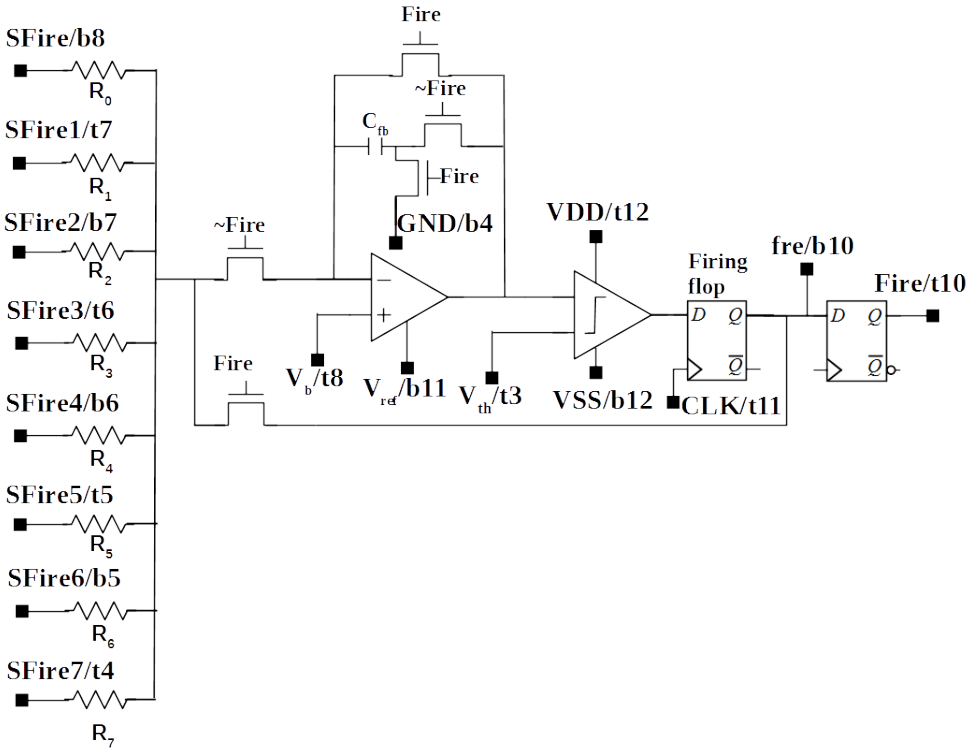

Figure B.9: Schematic for system prototype test structure B.3.

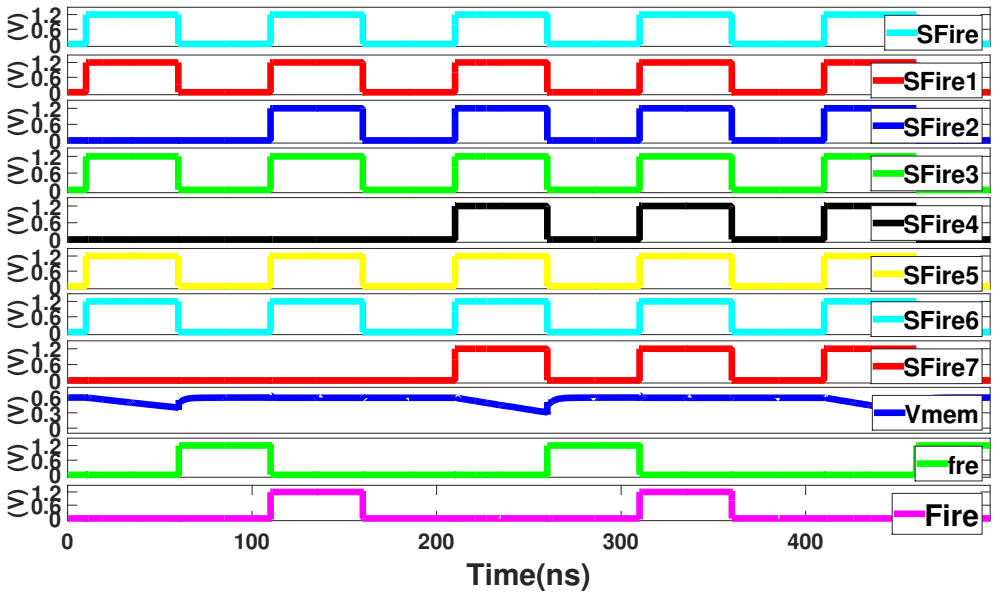

Figure B.10: Simulation of system prototype test structure B.3.

Vita

Gangotree Chakma received the B.Sc. degree in Electrical and Electronic Engineering from the Bangladesh University of Engineering and Technology, Dhaka, Bangladesh, in 2014. She joined the University of Tennessee, Knoxville in Fall 2015 for pursuing her Ph.D. degree in Electrical Engineering. Her research interest includes neuromorphic computing, mixed-signal circuit design, emerging nano-devices, and low-power VLSI designs. She is an active member of IEEE and ACM. She attended internship at Intel Corporation in 2018 and worked in machine learning for circuit-level efficiency. During her graduate studies, she worked as a member of UTK Neuromorphic group (TENNLab) and SENECA research group and helped in developing neuromorphic system for energy-efficient applications.